VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA A PREDIKCE Z GPS DAT

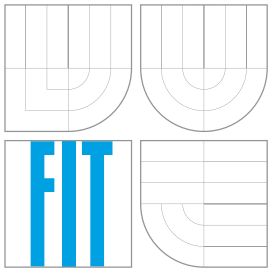
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DUŠAN KOVÁČIK

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ANALÝZA A PREDIKCE Z GPS DAT

ANALYSIS AND PREDICTION FROM GPS DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DUŠAN KOVÁČIK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2015

Abstrakt

Tato práce řeší analýzu sesbíraných GPS dat a na základě nich možnosti predikce nejvýhodnější trasy vypočítané za pomocí aplikace napsané ve skriptovacím jazyce PHP. Výhodnost trasy se posuzuje podle vzdálenosti, doby jízdy a převýšení. V práci je dále popsaný systém GPS, formát zdrojových dat a způsob jejich uložení do vhodné databáze. Nechybí ani popis hledání nejkratší cesty v grafu a několik nejznámějších algoritmů na její nalezení. Práce zahrnuje i popis implementace spracování nových dat a pozdější vyhledávání nad týmito datami ve skriptovacím jazyce PHP. V závěru je zhodnocený přínos této aplikace a návrh, jak je ji možné v budoucnosti vylepšit.

Abstract

This paper deals with the analysis of the collected GPS data and the possibility of the prediction of most advantageous route on its basis, by the application written in the PHP scripting language. The suitability of the route is considered by the distance, driving time, or elevation. The thesis also describes a GPS system, the format of the source data and their storing in an appropriate database. There is also a description of the search of the shortest path in the graph and some famous algorithms for finding it. The paper includes information about implementation of new data integration and path finding within this data in PHP scripting language. In conclusion it is evaluated what are the benefits of this application and design saying how this application can be improved in the future.

Klíčová slova

gps, teorie grafů, nejkratší cesta, predikce

Keywords

gps, graph theory, shortest path, prediction

Citace

Dušan Kováčik: Analýza a predikce z GPS dat, diplomová práce, Brno, FIT VUT v Brně, 2015

Analýza a predikce z GPS dat

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dušan Kováčik
26. května 2015

Poděkování

Děkuji svému vedoucímu práce Ing. Radku Burgetovi, Ph.D za odborné rady, ochotu a čas, který mi věnoval.

Obsah

Kapitola 1

Úvod

Vyhľadávanie informácií prostredníctvom internetu je dnes už každodennou praxou a učia sa mu deti na základných školách. Hľadať je možné hudbu, filmy, knihy, televízne programy, cestovné poriadky a mnoho ďalších viac či menej užitočných informácií. Každý z nás sa už určite niekedy potreboval dostať na určité miesto za čo najkratší čas, alebo ísť na dovolenku najkratšou cestou, aby ušetril peniaze na benzín. Na podobné vyhľadávanie dnes slúži množstvo webových aplikácii.

Problém môže nastať vtedy, keď chceme vyhľadať najrýchlejšiu trasu pre chodca. Každý má iné tempo chôdze, a preto je čas potrebný na prejdenie zvolenej cesty rôzny. Málokto sa však v dnešnej doby vyberie na cestu dlhšiu, než hodina(samozrejme až na turistov), takže odchýlka medzi najrýchlejším a najpomalším chodcom nie je až tak veľká.

Pri cyklistoch tento problém narastá ešte viac. Predsa len niekto dokáže zabrať do pedálov viac, než ostatní a problém mu nerobia ani väčšie prevýšenia. Tým môže byť časový výsledok pre dve takéto skupiny cyklistov príliš skreslený už po pár kilometroch. Najideálnejšie by bolo, keby si skupina profesionálnych alebo amatérskych cyklistov dokázala vyhľadávať trasy podľa vlastného tempa. Taktiež sa môže stať, že si chceme nájsť najkratšiu cestu za známimy na druhej strane kopca, no vyhľadávač nás pošle okľukou vôkol neho a my pritom vieme, že cez kopec síce vedie náročnejšia cesta, ale je tam.

Práve riešením týchto problémov sa zaoberá táto práca. V prvom kroku skupina cyklistov zaznamená svoj pohyb pomocou GPS zariadenia. Tento krok je splnený, nakoľko je k dispozícii záznam cyklistických výjazdov z takmer 10-tich rokov. Ďalej je potrebné navrhnúť aplikáciu, ktorá dokáže tieto údaje spracovať, uložiť ich do vhodnej databázy a umožniť užívateľom(cyklistom) nad týmito údajmi vyhľadávať. Administrátor navyše môže do databázy pridávať záznamy z ďalších výjazdov alebo ich prehliadať, či mazať.

V kapitole 2 je popísaný spôsob zaznamenávania pozícií pomocou GPS zariadenia vrátane jeho výhod a nevýhod. Nachádza sa tu aj popis niekoľkých formátov používaných pre ukladanie GPS záznamov vrátane toho zvoleného pre túto aplikáciu. Kapitola 3 poukazuje na problém vyhľadávania najvhodnejšej trasy. Je tu spomenutých pár základných pojmov z teórie grafov a niekoľko najčastejších vyhľadávacích algoritmov. Kapitola č. 4 popisuje návrh aplikácie. Nachádza sa tu zvolený implementačný jazyk, typ datbázy, ktorá bude uchovávať získané hodnoty, spôsob vyhľadávania najlepšej trasy, a popis možností v užívateľskom alebo administračnom rozhraní. Kapitola 5 popisuje použitý implementačný jazyk, aplikačné rozhranie umožňujúce výber bodov a zobrazenie výsledkov a popis implementácie pridávania nových trás a vyhľadávania nad nimi. Aplikácia, ktorá síce dáva pekné výsledky, ale neodráža realitu, je k ničomu. Testovanie je dôležité a práve tomu je venovaná kapitola 6. Práca končí kapitolou 7, kde je zhodnotený jej výsledok a naznačuje jej možné vylepšenie.

Kapitola 2

Spracovanie geografických dát

Táto kapitola je venovaná úvodnej problematike ohľadom vysielania a prijímania GPS signálu. Nasleduje popis dvoch známych formátov pre ukladanie záznamov o trase. Kapitola ďalej popisuje zdroj čerpaných dát a ich reprezentáciu. Druhá polovica kapitoly zmieňuje nerovnomerné rozprestrenie zaznamenaných pozícií a spôsob, akým bol tento problém riešený.

2.1 Technológia GPS

GPS, NAVSTAR - GPS (Navigation Satellite Timing and Ranging Global Positioning System) je navigačný systém, ktorým môžeme určiť svoju polohu kdekoľvek na zemskom povrchu bez ohľadu na počasie. GPS je pôvodne vojenský systém, vyvíjaný a budovaný od roku 1973 Ministerstvom obrany USA. Od roku 1983 je GPS využiteľný aj pre civilistov, i keď mal z dôvodu bezpečnosti zníženú presnosť pomocou umelej odchýlky. Tá znepresňovala pozíciu až o 100 metrov a odstránená bola v roku 2000. Od tej doby mohol začať hlavný rozvoj GPS prístrojov a navigácie. [12, s. 7].

Každá družica je vybavená prijímačom, vysielačom, atómovými hodinami(ktoré pracujú s presnosťou nanosekúnd) a radou prístrojov, ktoré slúžia pre navigáciu alebo iné špeciálne úlohy. Prijímače dokážu sledovať 8-12 družíc [12, s. 8]. Podľa [7] teoretická presnosť nemôže byť menšia, než 3 metre z dôvodu častého použitia nedostatočne presného elektronického detektora (vojenská technika disponuje s prístrojmi 10x presnejšími). V praxi sa však stretneme s faktom, že odchýlka sa pohybuje v okolí 10-tich metrov [9].

2.2 Formát GPX

GPX je formát využívajúci štandard XML slúžiaci pre presun zaznamenaných GPS súradníc medzi aplikáciami a webovými službami na internete [2]. Dokumenty GPX používajú koreňový element `<gpx>`, ktorý v úvode obsahuje hlavičku s metadatami, zoznam bodov záujmov, zoznam bodov trasy, zoznam bodov cesty a záver môže obsahovať prípadné rozšírenia.

Hlavička s metadátami sa nachádza v elemente `<metadata>` a obsahuje informácie ako meno autora súboru, dátum jeho vytvorenia, súradnice obdĺžnika ohraničujúceho zaznamenanú oblasť a ďalšie. **Body záujmov** sú umiestnené v elementoch `<wpt>`. Jeden záznam obsahuje názov bodu záujmu, jeho popis, GPS pozíciu, odkaz na ďalšie informácie a v závere je opäť možnosť pre rozšírenia. **Cesty** sú zoradeným zoznamom špeciálnych bodov na trase

vedúcej k cieľu a jeden záznam sa nachádza v elemente `<rte>`. Cesta opäť môže obsahovať názov a popis, externé odkazy a pod. Pre záznam samotnej prejdenej trasy je najdôležitejší **zoznam bodov trasy** nachádzajúci sa v elemente `<trk>` (v jednom súbore môže existovať viac ciest a každá bude v osobitnom elemente). Tento zoznam bodov môže obsahovať, podobne ako pri predošlých záznamoch, meno trasy, jej popis, externé odkazy, rozšírenia a ten najdôležitejší záznam – zoznam jednotlivých bodov v elemente `<trkseg>`. Každý bod sa potom nachádza v elemente `<trkpt>`, ktorý obsahuje minimálne GPS súradnice bodu, ale môže obsahovať aj nadmorskú výšku, čas záznamu, názov, popis a ďalšie, vrátane rozšírenia [1].

Tento formát je veľmi flexibilný, nakoľko umožňuje takmer pri každom zázname použiť vopred nedefinované rozšírenia. Tým môže byť napríklad pri ukladaní bodov na trase aktuálna teplota vzduchu.

2.3 Formát Keyhole Markup Language (KML)

Podobný formát využívaný vo veľkom aplikáciou Google Earth (GE) je formát KML. Podobne ako formát GPX, aj tento je založený na štandarde XML. Dokumenty v tomto formáte obsahujúce označenia miest, prekrytia zemského povrchu, cesty a polygony môžu byť vytvárané priamo zo spomínanej aplikácie [4].

Koreňovým elementom tohto formátu je `<kml>`. **Špeciálne značky** na trase sa ukladajú do elementov `<Placemark>`. Tie môžu obsahovať názov značky, jej popis, príznak, či má byť značka viditeľná a môže byť priradená k nejakému geometrickému prvku. Ak sa priradí napríklad k bodu, v GE sa zobrazí s ikonou. Od verzie KML 2.2 môžu tieto body obsahovať aj rôzne rozšírenia. Tento formát ďalej umožňuje **prekryť zemský povrch** určitým obrázkom. Táto možnosť sa zapisuje do elemntu `<GroundOverlay>`, ktorému sa môže priradiť názov, popis, súradnice prekrytia, natočenie, odkaz na samotný obrázok a ďalšie. **Záznam cesty** sa umiestňuje do elementu `<LineString>`. Ten obsahuje rôzne nastavenia pre zobrazenie cesty v aplikácii GE, no pre záznam trasy je dôležitý iba element `<coordinates>`, kde sú medzerami oddelené jednotlivé zaznamenané body. Jeden bod obsahuje zemepisnú dĺžku, zemepisnú šírku a prípadne nadmorskú výšku, kde tieto údaje sú oddelené čiarkami. Formát KML umožňuje dokonca vytvoriť jednoduché **3D modely**, ktoré sa neskôr zobrazia v GE. V dokumente sa ukladajú do elementov `<Polygon>` [3, 4].

Tento formát je možné použiť pre záznam cesty, avšak trpí nedostatkom možnosti ukladania času k zaznamenaným bodom a ďalších priebežných informáciách o ceste.

2.4 Zdroj dát a ich reprezentácia

Nakoľko výsledná aplikácia bude vyhľadávať cyklistické trasy, bolo žiaduce využiť databázu nejakého cyklistického združenia. Za najideálnejšie sa mi javilo použiť databázu združenia BIKELAND [5]. Ich webová stránka poskytuje možnosť stiahnuť trasy výjazdov, ktoré sa konajú v drvivej väčšine raz za jeden až dva týždne (väčšinou sú vynechávané len zimné mesiace, čo v posledných rokoch už tiež nie je pravda). Táto databáza by mala byť postačujúca, nakoľko len za rok 2014 disponuje trasami s celkovou vzdialenosťou takmer 3000 km a história výjazdov siaha až do roku 2005.

Webová stránka poskytuje záznam výjazdov vo formáte GPX vo verzii 1.1. Ide o textový formát XML, z ktorého je získanie údajov veľmi jednoduché. V globálnych metadátach sa nachádza obdĺžnik pokrývajúci všetky GPS body v danom výjazde. Umiestnený

je tu aj čas a určité zaujímavé body výjazdu, ktoré sú však pre vyhľadávanie trasy irelevantné. Najdôležitejšia sekcia súboru sa nachádza v XML tagu `<trk>`. Ten obsahuje jednotlivé GPS body výjazdu reprezentované zemepisnými súradnicami, nadmorskou výškou a časom záznamu. Pomocou týchto údajov je možné zistiť napríklad náročnosť terénu (podľa stúpania, či klesania) alebo rýchlosť jazdy [1].

2.5 Predpríprava získaných dát

Získané súradnice sú veľmi nerovnomerne rozložené, takže som sa rozhodol ich upraviť tak, aby bola medzi nimi približne rovnaká vzdialenosť. Túto vzdialenosť som nechcel dať veľmi vysokú, aby neboli ostré zákruty príliš zrezané a súbežné cesty nesplynuli v jednu. Zároveň táto vzdialenosť nemôže byť príliš malá, aby neskoršie vyhľadávanie v databáze zbytočne nezaťažila. S týmito požiadavkami som sa rozhodol určiť hodnotu vzdialenosti na 15 metrov. Podrobne je tomuto procesu venovaná sekcia 4.4.

Týmto spôsobom je možné presnejšie a rovnomernejšie určiť pozíciu GPS prístroja, aj keď ani pri najlepšom signále nie je väčšinou možné určiť pozíciu presnejšie, než na 3 metre [7]. Pri väčších vzdialenostiach je zas úsek rozdrobený na úseky s požadovanou vzdialenosťou.

Kapitola 3

Spôsoby vyhľadávania

V tejto kapitole sa venujem teórii grafov a spôsobu vyhľadávania najkratšej cesty. V úvode je spomenutých pár základných pojmov potrebných pre prácu s grafmi, a ktoré sú používané aj v ďalšej časti práce. Za týmto stručným úvodom nasleduje najjednoduchší spôsob vyhľadávania pomocou základného algoritmu. Ďalej nasleduje Dijkstrov algorimus, ktorý vyhľadáva optimalizovanejšie a spomenutá je aj jeho modifikácia s vyhľadávaním z oboch koncov súčasne. V závere kapitoly sa nachádza ešte algoritmus A*, ktorý sa vďaka pomoci vhodnej heuristickej funkcie zdá byť najvýhodnejší zo spomínaných.

3.1 Grafy

Podľa [8, s. 18] **grafom** nazveme usporiadanú dvojicu $G = (V, H)$, kde V je neprázdna konečná množina a H je množina neusporiadaných dvojíc typu $\{u, v\}$ takých, že $u \in V \wedge v \in V \wedge u \neq v$, čiže:

$$H \subseteq \{\{u, v\} \mid u \neq v \wedge u, v \in V\}$$

Príklad grafu sa nachádza na obrázku 3.1. Prvky množiny V sa nazývajú **vrcholy** a prvky množiny H **hrany**.

V [8, s. 18] sa spomína aj pojem **digraf**, čo je usporiadaná dvojica $\vec{G} = (V, H)$, kde V je neprázdna konečná množina a H je množina usporiadaných dvojíc typu (u, v) takých, že $u \in V \wedge v \in V \wedge u \neq v$, čiže:

$$H \subseteq \{(u, v) \mid u \neq v \wedge u, v \in V\}$$

Základný rozdiel medzi týmito dvoma grafmi spočíva v tom, že digraf môže obsahovať hrany platné len pre jeden smer. Prvky množiny H sa preto nazývajú **orientovanými hranami**.

Posledným základným pojmom z teórie grafov používaným v tejto práci je **cesta**. Tá je v [8, s. 59–60] definovaná ako striedajúca sa postupnosť vrcholov a hrán, v ktorej sa žiaden vrchol neopakuje. Cesta má v grafe tvar:

$$\mu(v_1, v_k) = (v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, v3, \ldots, v_{k-1}, \{v_{k-1}, v_k\}, v_k)$$

a v digrafe tvar:

$$\mu(v_1, v_k) = (v_1, (v_1, v_2), v_2, (v_2, v_3), v3, \ldots, v_{k-1}, (v_{k-1}, v_k), v_k)$$

3.2 Základný algoritmus

Základný algoritmus je založený na jednoduchom postupe, pričom pri kladne ohodnotených hranách vždy nájde najkratšiu cestu. Algoritmus je nasledovný:

- **Krok 1: Inicializácia**
  Majme množinu vrcholov V, množinu hrán (ciest) H, počiatočný vrchol $u \in V$ a koncový vrchol $v \in V$. Funkcia $c(x, y)$ udáva ohodnotenie orientovanej hrany z vrchola x do vrchola y. Každému vrcholu $i \in V$ priradíme značku $vzdialenost(i)$ a $predchodca(i)$. Značka $vzdialenost(i)$ predstavuje horný odhad dĺžky doteraz nájdenej najlepšej cesty z vrchola u do vrchola i a $predchodca(i)$ je značka predposledného vrchola v ceste. Polož $vzdialenost(u) := 0$, $vzdialenost(i) := \infty$, kde $i \in V \wedge i \neq u$ a $predchodca(i) := 0$ pre každé $i \in V$.

- **Krok 2:**
  Zisti, či existuje orientovaná hrana $(i, j) \in H$, pre ktorú platí:

  $$vzdialenost(j) > vzdialenost(i) + c(i, j)$$

  Ak taká hrana existuje, potom polož

  $$vzdialenost(j) := vzdialenost(i) + c(i, j)$$

  $$predchodca(j) = i$$

  a opakuj krok 2.

- **Krok 3:**
  Ak orientovaná hrana z kroku 2 neexistuje, najkratšiu cestu z vrcholu u do vrcholu i zostrojíme spätne pomocou značiek $predchodca(i)$ ako cestu prechádzajúcu vrcholmi

  $$i, predchodca(i), predchodca(predchodca(i)), \ldots, u$$

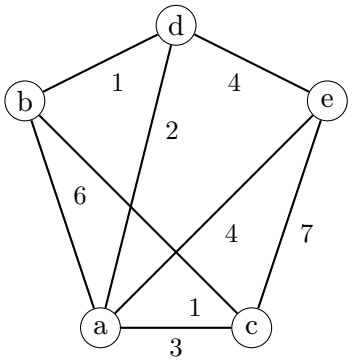
  STOP. [8, s. 72–73]

Pre názornosť uvažujme graf na obrázku 3.1, v ktorom sa chceme dostať najkratšou cestou z bodu e do bodu c. V inicializačnom kroku 0 nastavíme všetky vzdialenosti (okrem počiatočného vrcholu) na maximálnu vzdialenosť a všetkým vrcholom priradíme „nulového“ predchodcu. Vzdialenosti sú zaznačené v tabuľke 3.1 a predchodcovia v tabuľke 3.2.

V kroku 1 zistíme, že z vrchola e dokážeme nájsť kratšiu cestu, než doteraz uvedenú (maximálnu) vzdialenosť, do vrcholov a, c, a d. Vzdialenosti zaznačíme a zaznačíme aj predchodcu značených vrcholov, čo bude v tomto prípade vrchol e. V ďalšom kroku zistíme, že z vrchola a sa dá dostať do vrchola b kratšou cestou a v kroku 3, že do vrchola b sa dá dostať ešte kratšou cestou z vrchola c. Najkratšiu cestu do vrchola b nájdeme však až v kroku 4 a bude to z vrchola d. Na záver ešte zistíme, že do vrchola c sa dá dostať kratšou cestou z vrchola b, čo je zaznačené v kroku 5. V tomto stave sa už žiadna kratšia cesta nájsť nedá, čím algoritmus končí.

S tabuľkou predchodcov je teraz možné zostrojiť cestu. Vieme, že do vrchola c sa dostaneme z vrchola b, do ktorého sa dostaneme z vrchola d. Do neho prídeme z vrchola e, čo je vrchol, z ktorého sme hľadali cestu. Najkratšia cesta tak má tvar

$$e \rightarrow d \rightarrow b \rightarrow c$$



Obr. 3.1: Vzorový obrázok grafu.

	Krok 0	Krok 1	Krok 2	Krok 3	Krok 4	Krok 5
a	∞	4	4	4	4	4
b	∞	∞	10	8	5	5
c	∞	7	7	7	7	6
d	∞	4	4	4	4	4
e	0	0	0	0	0	0

Tabuľka 3.1: Priradené vzdialenosti jednotlivým vrcholom.

	Krok 0	Krok 1	Krok 2	Krok 3	Krok 4	Krok 5
a	0	*e*	*e*	*e*	*e*	*e*
b	0	0	*a*	*c*	*d*	*d*
c	0	*e*	*e*	*e*	*e*	*b*
d	0	*e*	*e*	*e*	*e*	*e*
e	0	0	0	0	0	0

Tabuľka 3.2: Zoznam predchodcov pri vstupe do daných vrcholov.

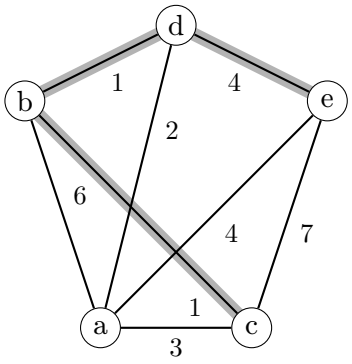
vyobrazený aj na obrázku 3.2.

Hlavnou nevýhodou tohto algortimu je jeho zložitosť. Pri n vrcholoch môže algoritmus vykonať až n^3 úkonov [8, s. 76–77]. V aplikácii s napríklad 1000 vrcholmi by to znamenalo až miliardu operácií, čo je nemysliteľné.

3.3 Dijkstrov algoritmus

Dijkstrov algoritmus predpokladá, že všetky hrany v grafe majú nezáporné ohodnotenie. Všetky cesty majú nezápornú vzdialenosť, nezápornú dobu jazdy medzi nimi a dokonca aj náročnosť terénu pri klesaní má nulovú hodnotu, takže tento algoritmus je možné použiť. Algoritmus vyhľadávania je nasledovný:

- **Krok 1: Inicializácia**
  Majme množinu vrcholov V, množinu hrán (ciest) H, počiatočný vrchol $u \in V$ a koncový vrchol $v \in V$. Funkcia $c(x, y)$ udáva ohodnotenie orientovanej hrany z vrchola x do vrchola y. Každému vrcholu $i \in V$ priradíme značku $vzdialenost(i)$ a $predchodca(i)$. Značka $vzdialenost(i)$ bude dvojakého druhu, a sa dočasná a definitívna. Polož $vzdialenost(u) := 0$, $vzdialenost(i) := \infty$, kde $i \in V \wedge i \neq u$



Obr. 3.2: Vzorový obrázok grafu s vyznačenou najkratšou cestou z vrchola e do vrchola c.

a $predchodca(i) := 0$ pre každé $i \in V$. Zvoľ riadiaci vrchol $r := u$ a značku $vzdialenost()$ pri tomto vrchole prehlás za definitívnu. Ostatné značky za dočasné.

- **Krok 2:**
  Ak je $r = v$, STOP. Ak $vzdialenost(v) < \infty$, značka $vzdialenost(v)$ predstavuje dĺžku najkratšej u-v cesty, ktorú zostroj podľa smerníkov $predchodca(i)$. Inak pre všetky hrany tvaru $(r, j) \in H$, kde j je vrchol s dočasnou značkou, urob:
  V prípade, že platí $vzdialenost(j) > vzdialenost(r) + c(r, j)$, potom $vzdialenost(j) := vzdialenost(r) + c(r, j)$, $predchodca(j) := r$ a ponechaj zmenené značky ako dočasné.

- **Krok 3:**
  Zo všetkých dočasne označených vrcholov nájdi ten vrchol i, ktorý má značku $vzdialenost(i)$ minimálnu. Značku pri tomto vrchole prehlás sa definitívnu a zvoľ nový riadiaci vrchol $r := i$.
  GOTO Krok 2. [8, s. 80]

Predstavme si opäť graf 3.1, v ktorom sa chceme dostať z vrcholu e do vrcholu d. Počiatočnému vrcholu v inicializačnom kroku priradíme honotu 0 a ostatným vrcholom hodnotu ∞. Tieto vzdialenosti sú značené v tabuľke 3.3. Predchodcovia vrcholov budú inicializovaní na hodnotu 0 a sú značení v tabuľke 3.4.

V kroku 1 má na začiatku najnižšiu vzdialenosť vrchol e, a tak ho prehlásime za riadiaci vrchol a jeho vzdialenosť 0 je definitívna. Presunom z vrcholu e dokážeme upraviť vzdialenosti vrcholov a, c a d, ktorých predchodcom bude vrchol e. V ďalšom kroku majú dva vrcholy rovnakú minimálnu vzdialenosť, takže si môžeme vybrať ľubovoľný z nich. Za riadiaci vrchol prehlásime vrchol a, ktorého dočasná značka sa zmení na definitívnu. Z neho môžeme vzdialenosť znížiť len vo vrchole b, ktorého predchodcom sa stane vrchol a. V kroku 3 sa stane riadiacim vrcholom vrchol d a dostane definitívnu značku. Ten opäť môže znížiť vzdialenosť len vo vrchole b, kde sa stane aj jeho predchodcom. V poslednom kroku bude riadiacim vrcholom vrchol b. Ten ešte dokáže zmeniť vzdialenosť v koncovom vrchole c a v ktorom sa stane predchodcom. V ďalšom kroku by bol riadiacim vrcholom vrchol c, ktorý je koncový, takže algoritmus by skončil. Zároveň je to však aj posledný vrchol s dočasnou značkou, a tak algoritmus končí.

Zložitosť tohto algoritmu je lepšia ako tomu bolo pri základom algoritme. Pri grafe s n vrcholmi bude výpočet trvať maximálne n^2 krokov [8, s. 80–81].

	Krok 0	Krok 1 $r = e$	Krok 2 $r = a$	Krok 3 $r = d$	Krok 4 $r = b$
a	∞	4	—	—	—
b	∞	∞	10	5	—
c	∞	7	7	7	6
d	∞	4	4	—	—
e	0	—	—	—	—

Tabuľka 3.3: Priradené vzdialenosti jednotlivým vrcholom.

	Krok 0	Krok 1 $r = e$	Krok 2 $r = a$	Krok 3 $r = d$	Krok 4 $r = b$
a	0	e	—	—	—
b	0	0	a	d	—
c	0	e	e	e	b
d	0	e	e	—	—
e	0	—	—	—	—

Tabuľka 3.4: Zoznam predchodcov pri vstupe do daných vrcholov.

3.4 Obojsmerný Dijkstrov algoritmus

Obojsmerný Dijkstrov algoritmus funguje rovnako, ako obyčajný Dijkstrov algoritmus, avšak cesta sa hľadá z oboch koncov. Algoritmus pre vyhľadávanie je nasledovný:

- **Krok 1: Inicializácia**
  Majme množinu vrcholov V, množinu hrán (ciest) H, počiatočný vrchol $u \in V$ a koncový vrchol $v \in V$. Funkcia $c(x, y)$ udáva ohodnotenie orientovanej hrany z vrchola x do vrchola y. Každému vrcholu $i \in V$ priradíme značku $vzdialenost(i)$ a $predchodca(i)$. Značka $vzdialenost(i)$ bude dvojakého druhu, a to dočasná a definitívna.
  Polož $vzdialenost_1(u) := 0$, $vzdialenost_1(i) := \infty$, kde $i \in V \wedge i \neq u$ a $predchodca_1(i) := 0$ pre každé $i \in V$. Polož $\mathcal{E}_1 := \{u\}$.
  Polož $vzdialenost_2(v) := 0$, $vzdialenost_2(i) := \infty$, kde $i \in V \wedge i \neq v$ a $predchodca_2(i) := 0$ pre každé $i \in V$. Polož $\mathcal{E}_2 := \{v\}$.

- **Krok 2:**
  Striedavo postupuj v priamom alebo opačnom smere podľa kroku 2a alebo 2b.

- **Krok 2a:**
  Vyber $r \in \mathcal{E}_1$ s najmenšou značkou $vzdialenost_1()$ a polož $\mathcal{E}_1 := \mathcal{E}_1 - \{r\}$. Značku $vzdialenost_1$ prehlás za definitívnu. Ak sú obe značky $vzdialenost_1(r)$ a $vzdialenost_2(r)$ definitívne, GOTO Krok 4. Inak pre všetky hrany $(r, j) \in H$ urob:
  Ak $vzdialenost_1(j) > vzdialenost_1(r) + c(r, j)$, potom
  $vzdialenost_1(j) := vzdialenost_1(r) + c(r, j)$, $predchodca_1(j) := r$, $\mathcal{E}_1 := \mathcal{E}_1 \cup \{j\}$.

- **Krok 2b:**
  Vyber $r \in \mathcal{E}_2$ s najmenšou značkou $vzdialenost_2()$ a polož $\mathcal{E}_2 := \mathcal{E}_2 - \{r\}$. Značku $vzdialenost_2$ prehlás za definitívnu. Ak sú obe značky $vzdialenost_1(r)$ a $vzdialenost_2(r)$ definitívne, GOTO Krok 4. Inak pre všetky hrany $(j, r) \in H$ urob:

Ak $vzdialenost_2(j) > vzdialenost_2(r) + c(j,r)$, potom
$vzdialenost_2(j) := vzdialenost_2(r) + c(j,r)$, $predchodca_2(j) := r$, $\mathcal{E}_2 := \mathcal{E}_2 \cup \{j\}$.

- **Krok 3:**
  Ak $\mathcal{E}_1 \neq \emptyset \wedge \mathcal{E}_2 \neq \emptyset$, GOTO Krok 2.
  Ak $\mathcal{E}_1 = \emptyset \vee \mathcal{E}_2 = \emptyset$, potom v je nedosiahnuteľný z u. STOP.

- **Krok 4:**
  Najkratšia u-v cesta vedie cez vrchol i, pre ktorý súčet značiek $vzdialenost_1(i)$ a $vzdialenost_2(i)$ je minimálny. Tieto značky pritom nemusia byť definitívne. Cesta vedie postupne cez vrcholy

  $$u, \ldots, pred_1(pred_1(i)), pred_1(i), i, pred_2(i), pred_2(pred_2(i)), \ldots, v$$

  kde *pred* je skratkou pre značku *predchodca*. [6, s. 27–28]

Algoritmus si ukážeme opäť na grafe z obrázku 3.1. Vzdialenosti sú zapísané v tabuľke 3.5 pri prehľadávaní v smere od začiatku a v tabuľke 3.6 pri prehľadávaní v smere od konca. Predchodcovia sa zas nachádzajú v tabuľke 3.7 vo vyhľadávanom smere a v tabuľke 3.8 v smere opačnom.

Na začiatok sa „vynulujú“ predchodcovia všetkých vrcholov a ich vzdialenosti sa nastavia na maximálnu možnú hodnotu. Akurát počiatočnému vrcholu v tabuľke 3.5 nastavíme nulovú hodnotu. Tú dáme aj koncovému vrcholu v tabuľke 3.6. Tým je inicializácia (Krok 0) dokončená.

V prvom kroku začneme vyhľadávať v smere od začiatku. Z množiny $\mathcal{E}_1$ vyberieme vrchol s najmenšou vzdialenosťou. Teraz je tam len jeden prvok, ktorý prehlásime za riadiaci vrchol. Z množiny $\mathcal{E}_1$ tento vrchol odoberieme a dáme mu definitívnu značku. Z riadiaceho vrchola dokážeme upraviť vzdialenosti vo vrcholoch a, c a d, pri ktorých nastavíme predchodcu vrchol e. Rovnako tieto vrcholy pridáme do množiny $\mathcal{E}_1$.

V kroku 2 sa presunieme do vyhľadávania od konca. Tentokrát z množiny $\mathcal{E}_2$ vyberieme opäť vrchol s minimálnou vzdialenosťou. Aj tu je zatiaľ len jeden prvok, ktorý z množiny odoberieme a prehlásime ho za riadiaci vrchol. Taktiež mu pridelíme definitívnu značku. Z neho dokážeme upraviť vzdialenosti vo vrcholoch a, b a d. Predchodcom týchto vrcholov sa stane vrchol c a vrcholy a, b a c pridáme do množiny $\mathcal{E}_2$.

Tretí krok začneme opäť výberom riadiaceho vrchola z množiny $\mathcal{E}_1$, kde hľadáme vrchol s najnižšou vzdialenosťou. Tým sa stáva vrchol a, ktorý z množiny odoberieme a pridáme mu definitívnu značku. Upraviť môžeme iba vrchol b, v ktorom je možné znížiť vzdialenosť, a ktorého predchodcom sa stane vrchol a. Vrchol b pridáme do množiny $\mathcal{E}_1$.

V štvrtom kroku vyberieme z množiny $\mathcal{E}_2$ vrchol b, ktorý v nej má najmenšiu vzdialenosť a stane sa riadiacim vrcholom. Rovnako dostane aj definitívnu značku. Z tohto vrchola môžeme upraviť vzdialenosť vrchola d, ktorý pridáme do množiny $\mathcal{E}_2$. Jeho predchodcom sa stane vrchol b.

V kroku 5 vyberieme tentokrát z množiny $\mathcal{E}_1$ vrchol d, pretože má teraz najmenšiu vzdialenosť, odoberieme ho z tejto množiny a pridáme mu definitívnu značku. Z vrchola d dokážeme skrátiť zatiaľ najväčšiu vzdialenosť vo vrchole b, kde sa stane aj jeho predchodcom.

Šiesty krok prebieha vo vyhľadávaní od konca. Množina $\mathcal{E}_2$ obsahuje vrcholy a, d a e. Z nich má najmenšiu vzdialenosť vrchol d. Ten však už má definitívnu značku v druhom vyhľadávaní, takže teraz je potrebné nájsť bod, cez ktorý sa nájdené najkratšie cesty spoja.

V tabuľke 3.9 sú súčty vzdialeností bodov z oboch smeroch vyhľadávania. Ako je možné vidieť, spojnicový bod môže byť vrchol b alebo vrchol d. Nezávisle na výbere sa dostaneme k zoznamu vrcholov, cez ktoré vedie najkratšia cesta. Tá je znázornená aj na obrázku 3.2.

	Krok 0 $\mathcal{E}_1 = \{e\}$	Krok 1 $r = e$ $\mathcal{E}_1 = \{a, c, d\}$	Krok 2 $\mathcal{E}_1 = \{a, c, d\}$	Krok 3 $r = a$ $\mathcal{E}_1 = \{b, c, d\}$	Krok 4 $\mathcal{E}_1 = \{b, c, d\}$	Krok 5 $r = d$ $\mathcal{E}_1 = \{b, c\}$
a	∞	4		—		—
b	∞	∞		10		5
c	∞	7		7		7
d	∞	4		4		—
e	0	—		—		—

Tabuľka 3.5: Priradené vzdialenosti jednotlivým vrcholom pri prehľadávaní od začiatku.

	Krok 0 $\mathcal{E}_2 = \{c\}$	Krok 1 $\mathcal{E}_2 = \{c\}$	Krok 2 $r = c$ $\mathcal{E}_2 = \{a, b, e\}$	Krok 3 $\mathcal{E}_2 = \{a, b, e\}$	Krok 4 $r = b$ $\mathcal{E}_2 = \{a, d, e\}$	Krok 5 $\mathcal{E}_2 = \{a, d, e\}$
a	∞		3		3	
b	∞		1		—	
c	∞		—		—	
d	∞		∞		2	
e	∞		7		7	

Tabuľka 3.6: Priradené vzdialenosti jednotlivým vrcholom pri prehľadávaní od konca.

	Krok 0	Krok 1 $r = e$	Krok 2	Krok 3 $r = a$	Krok 4	Krok 5 $r = d$
a	0	e		—		—
b	0	0		a		d
c	0	e		e		e
d	0	e		e		—
e	0	—		—		—

Tabuľka 3.7: Zoznam predchodcov pri vstupe do daných vrcholov pri prehľadávaní od začiatku.

	Krok 0	Krok 1	Krok 2 $r = c$	Krok 3	Krok 4 $r = b$	Krok 5
a	0		c		c	
b	0		c		—	
c	0		—		—	
d	0		0		b	
e	0		c		c	

Tabuľka 3.8: Zoznam predchodcov pri vstupe do daných vrcholov pri prehľadávaní od konca.

a	b	c	d	e
$4+3=7$	$1+5=6$	$7+0=7$	$4+2=6$	$0+7=7$

Tabuľka 3.9: Dĺžka cesty pri prechode jednotlivými bodmi

Táto modifikácia Dijkstrovho algoritmu síce nezmení jeho asymptotickú zložitosť, algoritmus je však približne 2x rýchlejší a prehľadá len polovičný počet hrán [6, s. 28–29]. Vyhľadávací priestor je znázornený na obrázku 3.3.

(a) Jednoduchý Dijkstrov algoritmus. (b) Obojsmerný Dijkstrov algoritmus.

Obr. 3.3: Porovnanie prehľadávaného priestoru pri oboch typoch Dijkstrových algoritmov.

3.5 Algoritmus A*

Podľa [6, s. 32–33] je princíp algoritmu A* nasledovný: Nech $f(i)$ je dĺžka najkratšej $u-v$ cesty idúcej cez vrchol i. Potom $f(u)$ je dĺžka najkratšej $u-v$ cesty (bez dodatočných podmienok). Ak vrchol i leží na najkratšej ceste, potom platí $f(i)=f(u)$. Naopak, ak i neleží na najkratšej $u-v$ ceste, potom $f(i)>f(u)$. Samozrejme hodnotu $f(i)$ nevieme jednoducho vypočítať (to je vlastne úloha, ktorú máme vyriešiť), preto algoritmus používa len jej odhad $\hat{f}(i)$.

f môžeme dekomponovať na $f(i)=g(i)+h(i)$, kde $g(i)$ je dĺžka najkratšej $u-i$ cesty a $h(i)$ je dĺžka najkratšej i–v cesty. Nech $\hat{g}(i)$ je odhad $g(i)$. Ako $\hat{g}(i)$ môžeme použiť dĺžku doteraz najlepšej nájdenej $u-i$ cesty v priebehu výpočtu. Samozrejme platí $\hat{g}(i)\geq g(i)$. Konštrukcia odhadu $\hat{h}(i)$ môže využiť doplňujúce údaje modelu. Jedným z dobrých odhadov je napríklad vzdušná vzdialenosť uzlov.

Ak by sme položili $\hat{h}(i)=0$ pre všetky $i\in V$, išlo by o dolný odhad dĺžky najkratšej $i-v$ cesty. V tomto prípade by išlo vlastne o Dijkstrov algoritmus.

Čím tesnejší dolný odhad $\hat{h}$ použijeme, tým menej vrcholov algoritmus prehľadá „zbytočne“, a bude teda pracovať efektívnejšie. Efektívnosť algoritmu preto úzko súvisí s kvalitou odhadu $\hat{h}$.

Asymptotická zložitosť algoritmu je rovnaká ako pri Dijkstrovom algoritme, čiže $O(n^2)$. Je však rýchlejší v absolútnom čase, keďže v prípade dobrého odhadu $\hat{h}$ prehľadá oveľa menej vrcholov digrafu.

Pri hľadaní cesty z vrchola u do vrchola v pomocou algoritmu A* možno zjednodušene postupovať nasledovne:

- **Krok 1: Inicializácia**
  Každý vrchol $i\in V$ dostane značku „neoznačený“ (ďalšie možné značky sú „otvorený“ alebo „uzatvorený“). Pre každý takýto vrchol je možné vypočítať hodnotu funkcie

$\hat{f}(i)$, určujúcu prioritu spracovania vrchola i. Spracovanie samotného vrchola možno všeobecne označiť ako aplikáciu operátora Γ na vrchol i.

- **Krok 2:**
  Označ vrchol u ako „otvorený“ a vypočítaj $\hat{f}(u)$.

- **Krok 3:**
  Vyber otvorený vrchol r, ktorý má minimálnu hodnotu $\hat{f}$.

- **Krok 4:**
  Ak $r = v$, označ r ako „uzatvorený“. STOP.

- **Krok 5:**
  Označ r ako „uzatvorený“ a aplikuj Γ na r. Pre každý vrchol $i \in V^+(n)$ vypočítaj hodnotu $\hat{f}(i)$ a ak je vrchol i „neoznačený“, preznač ho na „otvorený“. Ak už je vrchol i „uzatvorený“, označ ho ako „otvorený“ len v prípade, ak sa pre tento vrchol zmenšila hodnota $\hat{f}$. GOTO Krok 3. [6, s. 31]

Kapitola 4

Návrh

V tejto kapitole je naznačený vývoj aplikácie. V úvode je poukázaný cieľ aplikácie a možnosti, ktoré má ponúkať. Ďalej je spomenutý návrh databázy, v ktorej budú uložené všetky potrebné dáta pre vyhľadávanie. Potom nasleduje časť, ktorá detailne popisuje spôsoby spracovania zaznamenaných dát pomocou GPS prístroja a ich uloženie do databázy. V ďalšej časti tejto kapitoly sú popísané dva spôsoby vyhľadávania nad uloženými dátami a v závere sú naznačené užívateľské rozhrania pre bežného užívateľa a administrátora pridávajúceho nové dáta.

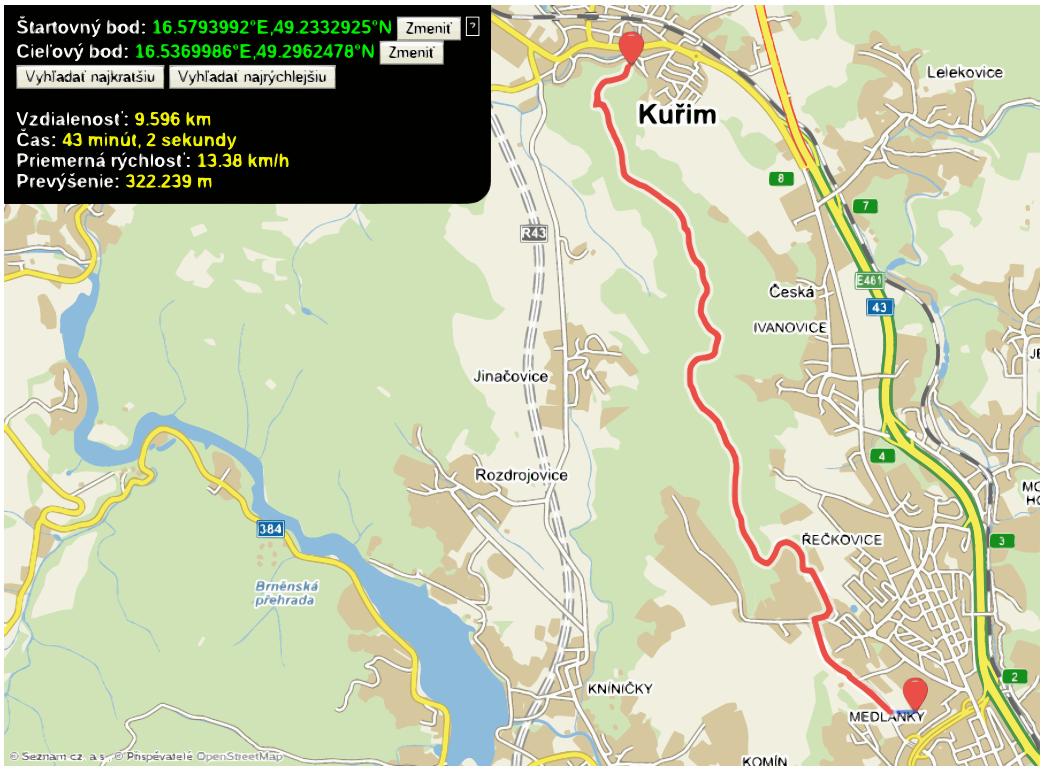
4.1 Možnosti výslednej aplikácie

Obr. 4.1: Diagram použitia navrhovanej aplikácie

Administrátor, ktorý chce používať aplikáciu na svojom serveri, by nemal byť nútený vytvárať databázu ručne, ale inicializovať ju jednoduchým spustením skriptu, ktorý vykoná všetku prácu za neho. Prázdna databáza by však nemala žiaden účinok, preto je nutné zabezpečiť možnosť administrátorovi pridávať nové trasy do systému. Administračné rozhranie bude dostupné len po zadaní hesla, ktoré bude vyžiadané po prvom prihlásení. V tomto rozhraní má administrátor možnosť pridávať nové trasy do systému, pomocou XML súboru s dátami vo formáte GPX 1.1. Spôsob spracovania a ukladania nových údajov je popísaný v sekciách 2.5 a 4.5.

Užívateľ, ktorý si spustí aplikáciu, musí mať možnosť zadať bod, z ktorého chce trasu vyhľadávať a bod, do ktorého sa chce dostať. Pre jednoduchosť by nemal byť nútený zadávať súradnice manuálne zápisom súradníc, ale výberom bodov na mape. Po zvolení týchto bodov si užívateľ môže zvoliť spôsob vyhľadávania tejto cesty a to buď spôsobom hľadania

najkratšej alebo najrýchlejšej cesty. Táto cesta by sa potom mala zobraziť na mape, kde sa zobrazia aj štatistiky o takejto ceste, ako je napríklad dĺžka trasy, prevýšenie a pod. Na obrázku 4.2 je ukážka takejto aplikácie.



Obr. 4.2: Vzhľad aplikácie pri vyhľadávaní ciest

4.2 Databáza

Už na začiatku bolo jasné, že GPS záznamov bude veľa, a že najvhodnejšia forma ich úložiska bude databáza. Na databázu som mal dve požiadavky – fukladanie geografických dát a jednoduchý prístup zo skriptovacieho jazyka php. Prieskumom existujúcich typov databáz som sa rozhodol použiť databázu MySQL. Tá spĺňa moje požiadavky, a navyše administračné rozhranie phpMyAdmin dokáže zobraziť uložené pozície na OpenStreetMap mape, čo je výhodou.

Databáza bude potrebovať tri základné tabuľky. Prvá z nich bude ukladať všetky získané a iným spôsobom upravené body a bude sa volať `points`. Každému bodu je potrebné priradiť minimálne jeho GPS pozíciu a nadmorskú výšku. V okolí bodov sa neskôr budú hladať ich blízki susedia, preto je vhodné všetky body rozdeliť do niekoľkých zón a každému bodu priradiť jednu vertikálnu a jednu horizontálnu zónu. Každý bod môže vystupovať ako uzol, alebo len ako bod na nejakej ceste, pomocou ktorého sa neskôr bude cesta vykresľovať. Na záver, každý bod má určitú váhu, ktorú je taktiež nutné ukladať. Ako bude neskôr ukázané (viď sekcia 4.5), z niekoľkých blízko umiestnených bodov môže vzniknúť nový bod, ktorého váha bude súčtom váh predchádzajúcich bodov. Táto váha bude ovplyvňovať ďalšie priemerovanie bodov.

Ďalšou potrebnou tabuľkou je tabuľka `ways`, ktorá obsahuje závislosti medzi uloženými

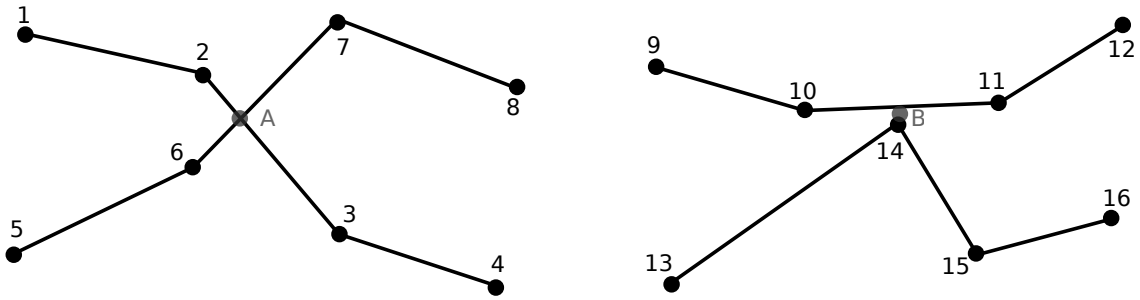
uzlami. Sú to hlavné hrany grafu, pomocou ktorých sa neskôr budú vyhľadávať cesty. Každý záznam bude obsahovať indexy bodov, ktoré spája, vzdialenosť medzi bodmi, čas a výšku potrebnú na prekonanie tejto vzdialenosti.

Poslednou potrebnou tabuľkou je tabuľka `connections`. Táto tabuľka určuje závislosť všetkých bodov medzi sebou. Pomocou každého takéto spojenia bodov musí byť možné určiť, ktoré body spája, ktorej ceste spojenie patrí a štatistiky spojené s takýmto spojením podobne ako pri cestách, teda vzdialenosť bodov, čas potrebný na prechod medzi nimi a výškový rozdiel.

Podrobnejšie je potreba ukladania týchto dát rozobratá v nasledujúcej sekcii. Čo sa týka databázy, je potrebné vytvoriť ešte jednu tabuľku, kam SQL skript uloží množinu blízkych bodov. Ten sa bude spúšťať zakaždým po pridaní nových bodov do databázy. Potreba získania takýchto bodov bude riešená v sekcii 4.5. Táto tabuľka bude obsahovať dva stĺpce a pracovať sa s ňou bude iba pri pridávaní novej cesty, kedy sa naplní, spracuje a opätovne vyprázdni. Praktickým príkladom takejto tabuľky je tabuľka 4.1.

4.3 Uchovávané dáta

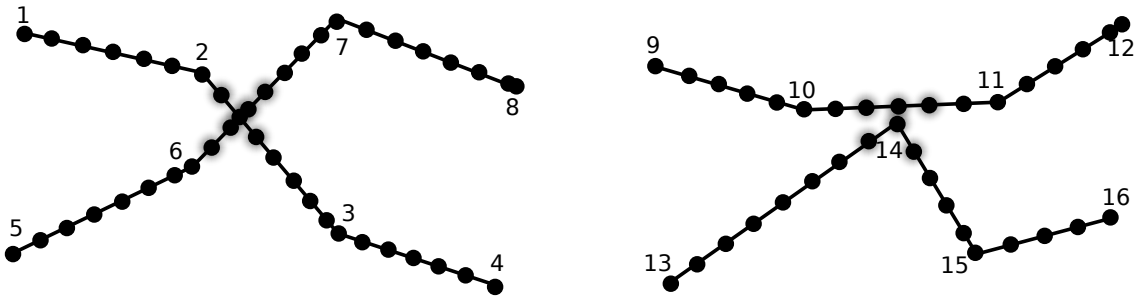
Veľkou chybou v prvom návrhu bolo domnievať sa, že postačujúce bude ukladať iba uzly, kde sa cesty krížia, a hrany obsahujúce informácie o vzdialenosti medzi uzlami. Pri pridávaní novej trasy do systému bolo však obtiažné nájsť oblasti, v ktorých majú uzly vzniknúť. Ako je možné vidieť na obrázku 4.3, uzly môžu vznikať dvoma spôsobmi. Uzol `A` vznikne prienikom hrán `2-3` a `6-7`. Uzol `B` môže vzniknúť v prípade, že sa vrchol `14` nachádza dostatočne blízko k hrane `10-11`. Z reálneho hľadiska si môžeme predstaviť dvoch cyklistov oproti sebe vchádzajúcich do križovatky, avšak obaja v nej zabočia vpravo. Aj keď sa ich cesty neskrížia, je žiaduce brať toto miesto ako uzol.



Obr. 4.3: Ukážka nie vždy jednoznačného prieniku

Riešením sa ukázalo rozdeliť zachytené úseky na čo najviac menších častí s rovnakou dĺžkou a neskôr len vyhľadávať body umiestnené vo svojej tesnej blízkosti. Pokiaľ sa takýto „zhluk bodov“ nájde, nahradí sa novým bodom, ktorého pozícia bude priemerom bodov v tomto „zhluku“. Zaznamenané GPS súradnice z meracieho zariadenia sú však často umiestnené buď príliš blízko seba, alebo až príliš ďaleko. Čo najrovnomernejšie rozprestrenie dát som rozobral v sekcii 4.4. Výsledný efekt je možné pozorovať na upravenom obrázku 4.4.

Nielen z tohto dôvodu je potrebné ukladať **všetky namerané body**, ich súradnice a váhu bodov pre neskoršie priemerovanie. Neskôr bude možné taktiež pomocou týchto bodov vykresliť finálnu vyhľadávanú trasu.



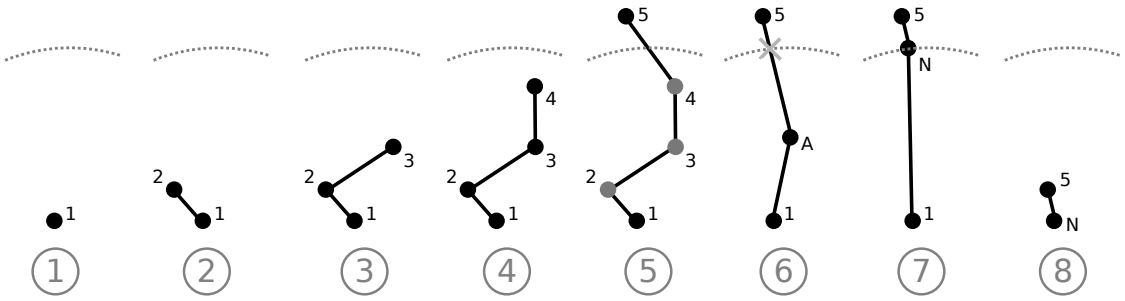
Obr. 4.4: Vylepšené vyhľadávanie prienikov ciest

Každý bod môže súvisieť s ďalšími bodmi a spoločne vytvárať **hrany**. Tie je taktiež potrebné ukladať, pretože so sebou nesú informácie o vzdialenosti bodov, čase potrebnom k presunu medzi nimi a prevýšení.

Tieto dáta by boli postačujúce ako pre neskoršie rozširovanie systému novými trasami, tak aj pre vyhľadávaní v takejto sieti. Vyhľadávanie by však prebiehalo nad príliš veľkým obnosom dát a trvalo by príliš dlho. Z tohto dôvodu si systém bude pamätať aj také **cesty** medzi uzlami, v ktorých sa nenachádza vetvenie. Budú obsahovať súčty dĺžkových, či časových vzdialeností a prevýšení hrán nachádzajúcich sa na ceste. Týchto ciest bude výrazne menej než hrán a bude sa nad nimi oveľa rýchlejšie vyhľadávať. Pre názornosť, miesto takmer 10 000 hrán sa použilo necelých 400 ciest.

4.4 Predspracovanie vkladaných dát do systému

Ako už bolo spomenuté, snímacie zariadenie GPS zaznamenáva pozície v nepravidelných intervaloch. To je však pre aplikáciu nevyhovujúce a zaznamenané body sú spriemerované a rozdelené na približne rovnaké intervaly. Na nasledujúcom obrázku 4.5 je znázornené, ako dochádza k priemerovaniu blízko umiestnených bodov.



Obr. 4.5: Priemerovanie získaných pozícií

Príklad 4.4.1 *Na začiatku priemerovania máme k dispozícii iba bod 1. V druhom kroku k nemu pridáme bod 2 a zistíme, že sa k prvému bodu nachádza príliš blízko. Bod je považovaný za príliš blízky, pokiaľ nepresiahne požadovaný rozostup medzi bodmi (znázornený bodkovaným oblúkom). V skutočnosti ide o 15 metrov, čo sa testovaním ukázalo ako rozumná hodnota. V krokoch 3 - 4 postupujeme podobne. V kroku 5 pridáme bod, ktorý už prekonal požadovanú hranicu. Tu sa zoberú všetky body v aktuálnom priemerovaní okrem prvého*

a posledného bodu. Spriemerujú sa teda body `2` až `4` a vznikne bod `A` zobrazený v kroku 6. Spriemerovaný bod `A` sa však nachádza príliš blízko a my chceme vytvoriť body s približne rovnakými rozostupmi. Úsečka `A-5` sa teda rozdelí v takom pomere, aby bol ďalší bod (bod `N` v nasledujúcom kroku) vzdialený od bodu `1` v požadovanej vzdialenosti. Bod `N` je teda výsledkom tohto priemerovania. V kroku 8 je naznačené, ako by priemerovanie pokračovalo ďalej. Body `N` a `5` sú predpripravené pre ďalšie priemerovanie, ktoré by pokračovalo obdobne, čím by opäť vznikli body s potrebným rozostupom.

Mohla by ale nastať aj taká situácia, že by bol bod `5` vzdialený príliš ďaleko už v kroku 8. V takomto prípade by priemerovanie nenastalo a úsečka `N-5` by sa rovno rozdelila v požadovanej vzdialenosti.

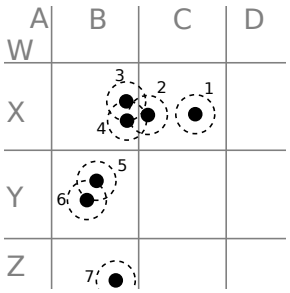
Graficky je poukázané na priemerovanie GPS súradníc, ale pri skutočnom priemerovaní treba brať do úvahy aj priemerovanie časových vzdialeností medzi bodmi a rovnako aj ich prevýšenie. Časový rozdiel sa počíta tak, že sa najprv zistí, akými rýchlosťami sa medzi jednotlivými úsekmi prešlo, tie sa spriemerujú a čas nového úseku bude získaný na základe vzdialenosti a spriemerovanej rýchlosti nového úseku.

4.5 Pridanie novej cesty

Pri vytváraní novej cesty sa už pracuje s bodmi, ktoré prešli priemerovaním popísaným v predošlej sekcii. Všetky tieto body je teraz potrebné pridať do databázy. Taktiež sa tam uložia všetky hrany medzi bodmi a aj novovzniknutá cesta, ktorá smeruje od prvého bodu po posledný.

Takáto cesta však ešte nespolupracuje s ostatnými dátami uloženými v systéme. Ďalším krokom preto je nájsť body, ktoré sa objavia vo svojej tesnej blízkosti. Databáza ale veľmi rýchlo naberá na počte uložených bodov, takže nie je možné prejsť vzdialenosťami medzi všetkými bodmi. Tie sú preto rozdelené do zón a pri vyhľadávaní blízko umiestnených bodov stačí prejsť aktuálnu zónu a 8 zón vôkol. Prečo nestačí prejsť len aktuálnu zónu ukážem teraz.

Príklad 4.5.1 *Na obrázku 4.6 je znázornených 16 zón. Keby som vyhľadával len v jednej zóne, našiel by som len dvojicu bodov `3-4` a `5-6`. Dvojice `2-3` a `2-4` sa taktiež nachádzajú blízko seba, avšak tie by mi inak unikli. Ďalej je potrebné vyhnúť sa duplicitám záznamov. Bod `2` je v blízkosti bodu `3` a rovnako je bod `3` v blízkosti bodu `2`.*



Vyhľadávanie z bodu	Nájdené blízke body
1	
2	3, 4
3	4
4	
5	6
6	
7	

Obr. 4.6: Vyhľadávanie blízkych bodov

Aby spracovanie blízkych bodov prebehlo čo najrýchlejšie, je žiaduce, aby bolo záznamov čo najmenej, avšak nič potrebné neuniklo. Vyhľadávať preto budem vždy len body s vyšším indexom než má bod, od ktorého sa blízke body vyhľadávajú.

Každá dvojica blízkych bodov je kandidátom na budúci uzol, nakoľko sa tieto body zväčša nachádzajú na hranách rôznych ciest. To však ale nie je podmienka a výsledok priemerovania bodov musí naprv prejsť niekoľkými krokmi.

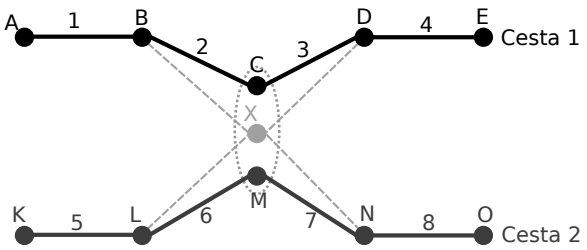
V prvom rade teda získam body, ktoré sa objavili blízko seba. Ďalším krokom je získať zoznam hrán, v ktorých sa tieto body vyskytujú. Podobne je potrebné získať aj zoznam ciest. Blízke body sú interpretované vždy dvojicou (P, PS), kde P je bod, ku ktorému je množina bodov PS umiestnená príliš blízko.

Súradnice bodov v množine $NPS = \{P\} \cup PS$ sa teraz spriemerujú a na ich mieste vznikne nový bod AP. Tento bod sa zatiaľ nevyskytuje v žiadnej hrane. Teraz je potrebné prejsť postupne všetkými bodmi v množine NPS. Jeden takýto bod budem označovať NP.

Pokiaľ bod NP predtým nebol označený ako uzol, je nutné vziať množinu ciest WS_{NP}, v ktorých sa bod NP vyskytoval. Tieto cesty totiž budú teraz predelené novým bodom AP a nebudú môcť existovať v celku. Pokiaľ sa však bod NP vyskytoval na začiatku alebo konci pôvodnej cesty, stačí len zaktualizovať pôvodnú cestu. Rovnako, ak bol bod NP predtým považovaný za uzol, žiadne nové cesty nevzniknú, len sa zaktualizuje pôvodná. Cesty sú vždy vedené od uzla k uzlu, takže tu je postačujúce preznačiť pôvodný bod za nový a zaktualizovať štatistiky o ceste. Treba ale dávať pozor na to, aby sa nestalo, že všetky priemerované body tvorili jednu cestu. Takáto cesta totižto zanikne.

Ďalej je nutné zaktualizovať hrany vedúce z alebo do bodu NP. Tu už nie je potrebné rozlišovať, či bol pôvodný bod uzol, pretože pri všetkých hranách dochádza len k preznačeniu pôvodného bodu v hrane za nový. Komplikovanejšie je to však s aktualizáciou ciest, ku ktorým hrany patria. Preznačovať sa totiž musia všetky hrany na pôvodnej ceste a nie iba tie, ktoré obsahovali pôvodný bod.

Príklad 4.5.2 *Než budeme pokračovať, predstavme si situáciu na obrázku 4.7. Nachádzajú sa tu dva blízko seba umiestnené body. Množina blízkych bodov by teda mala len jeden prvok – $\{(C, \{M\})\}$. Nastalo by preto iba jedno zlučovanie bodov. Množina NPS by obsahovala prvky $\{C, M\}$. Tieto dva body sa spriemerujú a vznikne nový bod X. Pri mazaní bodu C je potrebné najprv preznačiť hrany, v ktorých sa nachádzal, tak, aby hrany viedli do nového bodu. Z hrán $B - C$ a $C - D$ sa stanú hrany $B - X$ a $X - D$. Cestu 1 je potom nutné rozdeliť na dve, keďže bod C nebol uzlom. Z cesty $A - E$ vzniknú nové cesty $A - X$ a $X - E$ a všetkým hranám na týchto cestách je nutné zaktualizovať index cesty, ku ktorej od teraz patria. Hrany 1 a 2 sa pridelia prvej z nich a hrany 3 a 4 druhej. Obdobne sa vykonajú zmeny aj pri mazaní bodu M.*



Obr. 4.7: Spojenie bodov blízkych ciest

Hrany a cesty by týmto boli zauktualizované. Práca však zďaleka nie je hotová. Predchádzajúci príklad bol najjednoduchším prípadom, ktorý môže nastať.

Príklad 4.5.3 *Na obrázku 4.8 sú hrany značené arabskými číslicami, cesty rímskymi a vrcholy písmenami abecedy. V ľavej časti obrázku vidíme pôvodný stav grafu s troma cestami, ktoré sa spájajú vo vrchole C. Predstavme si ale, že vrcholy D a F sa nachádzajú blízko seba a budú sa zlučovať. Tým vznikne nový bod H, ktorý rozdelí pôvodné cesty II a III na dve časti, čím vzniknú nové cesty IV a V. Takýmto spôsobom však vzniknú medzi bodmi C a H dve rôzne cesty, čo je nežiaduce.*

Je preto nutné po preznačení hrán a rozdelení ciest skontrolovať, či sa medzi spriemerovaným bodom a jeho „susedom“ nenachádzajú duplicitné cesty. Ak sa nájdu, je potrebné ohodnotenia týchto hrán spriemerovať a nahradiť ich novou hranou a cestou.

Obr. 4.8: Vznik duplicitných ciest a hrán

Na obrázku 4.6 bolo naznačené, ako môže vypadať tabuľka blízkych bodov. Aj pri tej môže nastať niekoľko komplikácii, ktoré je potrebné kontrolovať a včas riešiť.

Príklad 4.5.4 *Tabuľka 4.1 obsahuje zoznam blízkych bodov. Z prvého riadku sa môžeme dočítať, že v blízkosti bodu `1` sa nachádzajú body `3` a `4`. Tieto body sa spriemerujú, čím vznikne nový bod s označením napríklad `7`. Ako už bolo skôr spomenuté, po spriemerovaní určitej množiny bodov budú body v tejto množine zmazané. Keby sme teraz chceli pokračovať, tak už pri druhom priemerovaní by sme dospeli k chybe, pretože bod `3` v tom čase už existovať nebude.*

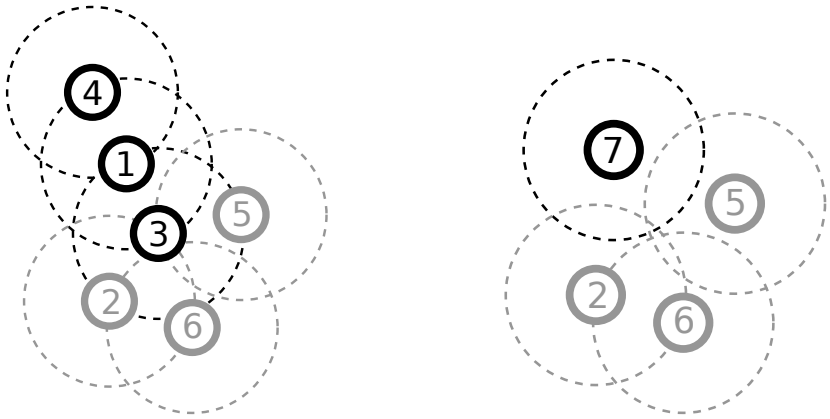
Vyhľadávanie z bodu	Nájdené blízke body
1	3, 4
2	3, 6
3	5, 6
4	
5	6
6	

Tabuľka 4.1: Príklad tabuľky blízkych bodov

Na základe predošlého príkladu by sa dalo predpokladať, že po každom priemerovaní bodov by sa mali prejsť všetky záznamy v tabuľke blízkych bodov a indexy zmazaných bodov nahradiť indexom nového spriemerovaného bodu. Z časti je to správne riešenie, ale nasledujúci príklad poukáže na jeden nedostatok.

Príklad 4.5.5 *Použijeme opäť tabuľku blízkych bodov 4.1. Graficky by mohli byť tieto body rozmiestnené podobne, ako je tomu na obrázku 4.9a. Spriemerovaním bodov `1`, `3` a `4` dostaneme bod `7`, čo je naznačené na obrázku 4.9b. Prvý záznam z tabuľky blízkych bodov je teraz*

možné zmazať. Ďalší výskyt bodov `3` a `4` je potrebné nahradiť bodom `7`. Už na novom prvom riadku tabuľky blízkych bodov (pôvodný prvý je zmazaný) je vidieť, že je potrebné nahradiť bod `3` za bod `7`. Keď sa ale opätovne pozrieme na obrázok 4.9b, tak zistíme, že bod `2` nie je v blízkosti bodu `7`.



(a) Pred spracovaním prvého záznamu (b) Po spracovaní prvého záznamu

Obr. 4.9: Priemerovanie bodov z tabuľky 4.1

Pri aktualizácii indexov bodov v tabuľke blízkych bodov je teda nutné pri každom nahradzovaní indexov skontrolovať, či sú tieto indexy v tabuľke stále platné.

Po spracovaní jednej dvojice (P, PS) je potrebné skontrolovať všetky body v okolí nového bodu AP. Môže sa totiž stať, že tieto body stratili svoj príznak uzla, alebo ho naopak získali.

Bodu má patriť príznak uzla vtedy, pokiaľ sa v tomto bode nevetvia cesty, alebo sa nachádza na jej začiatku alebo konci. Tento príznak zas bod nemá mať, ak doňho vstupuje jedna hrana a jedna vychadza. To je prípad jednosmerného priechodu bodom. Bod však nie je uzlom ani v takom prípade, ak sa z neho síce možno dostať do dvoch rôznych bodov (je tu vetvenie), ale z týchto bodov sa možno dostať späť. To je príklad obojsmerného priechodu bodom, čiže v konečnom dôsledku nejde o vetvenie ciest.

Príklad 4.5.6 *Jeden príklad situácie, kedy nejaký bod, ktorý má príznak uzla, môže o tento príznak prísť, je načatý na obrázku 4.8. Tam po spriemerovaní hrán `3` a `5` vznikne jedna hrana. Bod `C` bol predtým uzlom, ale teraz doňho vedie len jedna hrana a jedna hrana z neho vychádza. Ako bolo spomenuté, takýto bod uzlom byť nemá.*

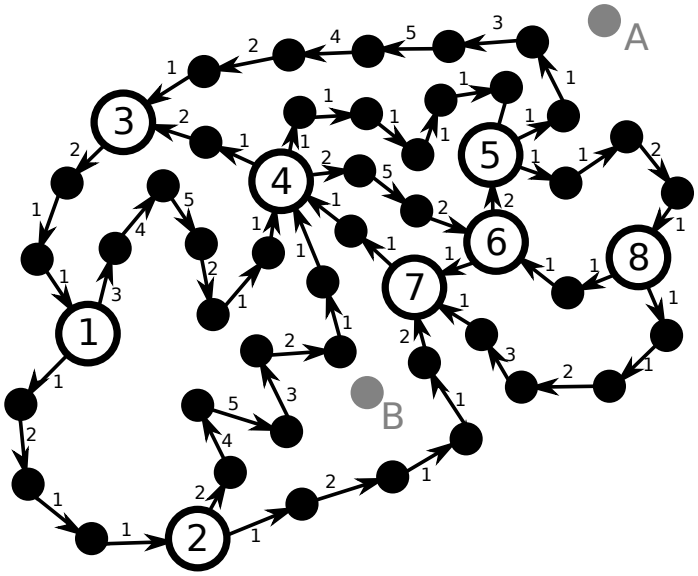
Teraz je už možné bezpečne zmazať body z množiny NPS, pretože sú všetky hrany a cesty presunuté do body AP. Ak sa v tabuľke blízkych bodov nachádza ďalší záznam, tak sa z nej prvý vyberie a celá popísaná akcia sa zopakuje. Ak je tabuľka prázdna, je možné do databázy nahrať všetky vykonané zmeny.

4.6 Hľadanie najlepšej trasy

Hľadanie najvhodnejšej trasy sa bude opierať o poznatky z teórie grafov uvedené v kapitole 3. Najlepšia trasa sa bude hľadať ako najkratšia cesta v grafe, pričom hrany budú ohodnotené vzdialenosťou vrcholov (hľadanie najkratšej trasy), dobou jazdy medzi vrcholmi (hľadanie najrýchlejšej trasy), a prevýšením (zobrazenie výsledného prevýšenia).

Vrcholy takéhoto grafu budú vznikať ako začiatky, konce a kríženia trás zadaných do systému, a hrany vzniknú na základe známych údajov o jazde medzi týmito vrcholmi. Vyhľadávať sa bude pomocou algoritmu A* (viď sekcia 3.5). Pri vyhľadávaní najkratšej cesty sa však bude zanedbávať orientácia hrán, nakoľko vzdialenosť medzi dvoma bodmi je vždy rovnaká. Naopak pri vyhľadávaní najrýchlejšej cesty sa bude vyhľadávať iba v zaznamenanom smere, pretože napríklad jazda do kopca zvyčajne zaberie viac času, než jazda z kopca.

Keď bude chcieť užívateľ vyhľadať cestu medzi dvoma bodmi, je veľká šanca, že netrafí presné miesto, kde boli nejaké dáta zaznamenané. Taktiež sa môže stať, že dáta nebudú zaznamenané ani v najbližšom okolí. V prvom rade je preto potrebné vyhľadať najbližšie známe pozície k zvoleným bodom. Tieto body sa potom stanú dočasnými uzlami, pretože v grafe sa budú vyhľadávať iba cesty medzi uzlami a nie medzi všetkými bodmi.

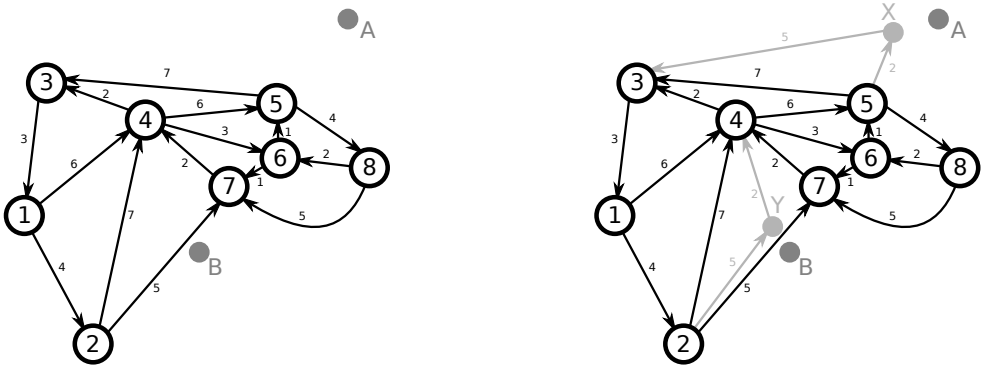


Obr. 4.10: Príklad grafu pre vyhľadávanie ciest so všetkými bodmi

Príklad 4.6.1 *Majme graf na obrázku 4.10, v ktorom budeme hľadať nakratšiu cestu z bodu* `A` *do bodu* `B`*. Hrany sú ohodnotené časom potrebným k prejdeniu z jedného bodu do druhého. Predpokladajme, že vzdialenosti medzi bodmi sú rovnaké. Na základe počtu bodov na cestách môžeme tento graf prekresliť na tvar zobrazený na obrázku 4.11a, kde sú hrany ohodnotené práve podľa vzdialeností. Ako bolo spomenuté, prvým krokom vyhľadávania ciest je nájdenie najbližších bodov k zvoleným bodom* `A` *a* `B`*, čo sú body* `X` *a* `Y`*. Tieto dva body sú pridané do grafu a pridajú sa aj hrany spájajúce tieto dva body s uzlami, ktorými sú ohraničené cesty, na ktorých sa body* `X` *a* `Y` *nachádzajú. Ukážkou výsledného efektu je obrázok 4.11b.*

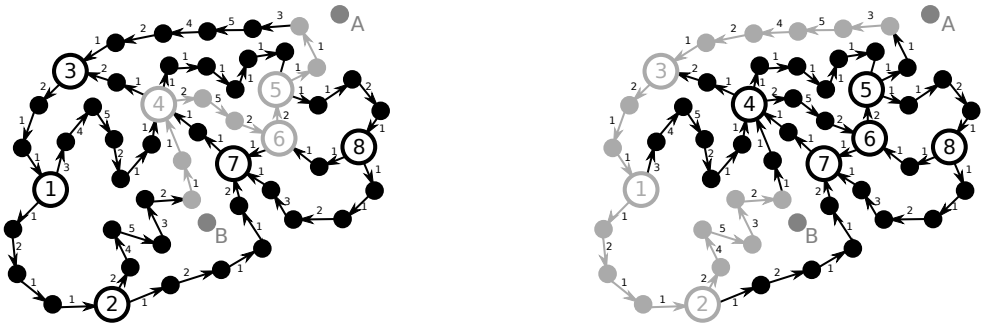
Vyhľadávanie iba pomocou uzlov je výrazne rýchlejšie, než na základe všetkých bodov. Po vyhľadaní cesty je žiaduce vykresliť celú cestu, preto je potrebné získať súradnice všetkých bodov na cestách, po ktorých bola najkratšia, či najrýchlejšia cesta nájdená. Výsledkom predchádzajúceho príkladu by bola cesta znázornená na obrázku 4.12a. Keby sa vyhľadávala najrýchlejšia cesta, je potrebné vziať do úvahy aj orientáciu hrán, preto by výsledok vypadal podobne, ako je tomu na obrázku 4.12b.

Vyhľadávanie trás som nechcel limitovať len na systém, ktorý som vytvoril. Oddelil som preto vrstvu pre vyhľadávanie najlepšej cesty od vrstvy, ktorá túto cestu zobrazí. Hocikto,



(a) Uzly bez modifikácie (b) Pridanie pomocných uzlov

Obr. 4.11: Príklad grafu pre vyhľadávanie najkratšej cesty s uzlami



(a) Najkratšia cesta (b) Najrýchlejšia cesta

Obr. 4.12: Vyhľadané cesty medzi bodmi `A` a `B`

kto bude chcieť získať dáta o najlepšej ceste medzi dvoma bodmi, pošle systému súradnice týchto dvoch bodov a systém mu vráti súradnice cesty, jej dĺžku a ďalšie štatistiky.

Kapitola 5

Implementácia

Táto kapitola rozoberá problematiku implementácie návrhu popísaného v predchádzajúcej kapitole. Na začiatku kapitoly je spomenutý výber programovacieho jazyka, pomocou ktorého bude aplikácia spracovávať údaje nových trás a zároveň tieto trasy vyhľadávať. Ďalej nasleduje popis spôsobu implementácie pridávania novej trasy vrátane ukážok riešenia komplikovanejších problémov. Aplikácia spočiatku bežala príliš pomaly, a preto sa v tejto kapitole nachádza aj popis skrátenia doby komunikácie s databázou. Záver je venovaný implementácii vyhľadávania trás a vytvoreniu užívateľského rozhrania pre toto vyhľadávanie a zobrazenie výsledkov.

5.1 Skriptovací jazyk PHP

PHP je často používaný open-source skriptovací jazyk, ktorý je špeciálne zameraný na tvorbu webových stránok a môže byť vložený do kódu HTML [11].

Výsledná aplikácia bude napísaná práve v tomto jazyku nielen z dôvodu jeho jednoduchosti, ale aj vďaka veľkej komunite okolo neho, čo často pomôže rýchlo riešiť vzniknuté problémy. Taktiež podporuje priamu komunikáciu s niekoľkými typmi databáz a prácu so súbormi (vrátane ukladania užívateľských súborov na server).

5.2 Mapy.cz API

Toto API umožňuje interaktívnu prácu s mapami z portálu `www.mapy.cz`. Pomocou neho je možné vybrať počiatočný či cieľový bod, zobraziť nájdenú najvhodnejšiu trasu a dostupných je niekoľko ďalších funkcií popísaných v [10]. Výhodou je jednoduchá implementácia s využitím skriptovacieho jazyka JavaScript, ktorý podporuje technológiu AJAX, a teda pri vyhľadávaní nie je nutné opätovne načítavať stránku.

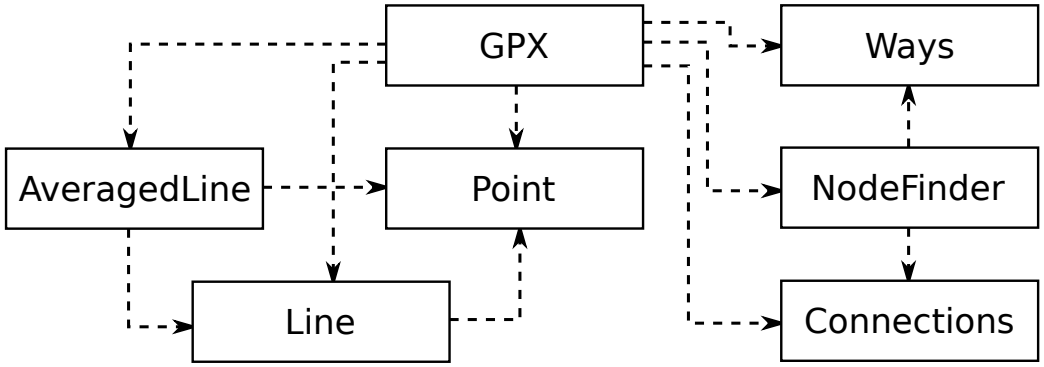
5.3 Vytvorenie novej databázy

Pokiaľ chce užívateľ vytvoriť na svojom serveri vlastný vyhľadávač, musí najprv spustiť súbor `/admin.php`. Tu zadá adresu servera, názov databázy (tá už musí existovať), meno a heslo užívateľa, ktorý má k danej databáze prístup.

Skript vytvorí požadované tabuľky, funkcie a procedúry potrebné k správnej funkčnosti systému. Pokiaľ nebude možné databázu pripraviť, užívateľ bude o tom informovaný.

5.4 Pridanie novej trasy do systému

Nové trasy do systému je možné pridať spustením súboru `/newData.php`. Na začiatku je vyžiadané heslo, ktoré užívateľ zadá pri prvom prihlásení. Zmeniť heslo je potom možné zmazaním súboru `password` na serveri. Po prihlásení má užívateľ možnosť zvoliť súbor s dátami vo formáte GPX v1.1. Tento formát má výhodu v tom, že v podstate ide o súbor vo formáte XML a na jeho rozparserovanie je možné použiť PHP funkciu `simplexml_load_file`. Objekt s týmito dátami je ďalej posunutý objektu `Gpx`, ktorý ich spracuje ďalej.



Obr. 5.1: Závislosť tried pri pridávaní novej trasy

Objekt `Gpx` pochopiteľne v prvom rade načíta zaznamenané body, na čo slúži metóda `getTrackPoints`. Táto metóda skontroluje formát vstupného súboru a vytvorí pole bodov načítaných z tohto súbora. Každý bod reprezentuje jedna inštancia objektu `Point`. Tento objekt obsahuje GPS pozíciu bodu, jeho nadmorskú výšku, časové razítko a váhu. GPS pozícia a nadmorská výška sa predáva konštruktoru objektu a neskôr tieto údaje nie je možné v rámci jednej inštancie zmeniť.

Body je potom potrebné rozmiestniť do približne rovnako vzdialených intervalov (viď sekcia 4.4). Na takéto rozmiestňovanie bodov bol vytvorený pomocný objekt `AveragedLine`. Tomu sa pomocou metódy `AveragedLine::setMaxLength` nastaví požadovaný rozostup medzi bodmi. Tomuto objektu sa potom už len cyklicky pridávajú jednotlivé body pomocou metódy `AveragedLine::addPoint`, avšak pred každým pridaním nového bodu je potrebné skontrolovať, či už rozostup medzi bodmi nebol prekročený. Na to slúži metóda `AveragedLine::isLengthExceeded`, ktorá zistí, či nie je posledný bod vzdialený príliš ďaleko od prvého bodu. Ak by táto metóda vrátila pravdivý výsledok, je nutné vyžiadať si bod, ktorý bude výsledkom priemerovania. Na to je potrebné najprv zavolať metódu `AveragedLine::breakAtMaxPosition`, ktorá spriemeruje získané body a vytvorí novú inštanciu objektu `AveragedLine`, ktorá bude obsahovať preddefinované body pre ďalšie priemerovanie. Táto situácia je vyobrazená na obrázku 4.5 v kroku 8. Pozíciu samotného spriemerovaného bodu je potom možné vyžiadať pomocou metódy `AveragedLine::getEndLinePoint`. Celé toto priemerovanie vykonáva metóda `Gpx::createAveragedPoints`.

Tieto body sa teraz uložia do databázy. Metóda `Gpx::savePoints` vytvorí jeden SQL príkaz, ktorý pomocou jedinej požiadavky na databázu vytvorí všetky nové body. Keby sa každý bod ukladal do databázy osobitnou požiadavkou, trvalo by to oveľa dlhšie. Ukladanie približne 1 000 bodov sa podarilo týmto skrátiť z 15 sekúnd na 2. Po uložení týchto bodov sa ešte vyžiada z databázy index prvého vloženého záznamu. Ten bude o chvíľu potrebný, preto bude návratovou hodnotou tejto metódy.

Pole bodov je momentálne indexované od nuly. Rýchlejší prístup k bodom by však umožnili indexy bodov odpovedajúce tým v databáze. Korekciu týchto indexov vykoná metóda `Gpx::correctPointIndexes`, ktorej parametrom je práve index získaný z predchádzajúcej operácie.

Ďalším krokom je vytvorenie inštancie objektu `Ways`. Tento objekt obsahuje určitú množinu ciest, ktorú spravuje – načítavá údaje z databázy, modifikuje ich, pridáva nové cesty a ukladá ich späť do systému. Teraz je potrebné vytvoriť novú cestu, na čo je určiená metóda `Ways::createNewWay`. Parametrami sa určia indexy prvého a posledného bodu. Návratovou hodnotou je index novej cesty, ktorý sa použije pri vytváraní spojení medzi jednotlivými bodmi.

Spojenia medzi bodmi vytvorí metóda `Gpx::createConnectionsFromPoints`, ktorá parametrom dostane množinu správne naindexovaných nových bodov. Najprv sa postupne prejde všetkými bodmi (tie sú momentálne zoradené, takže každé dva po sebe idúce body tvoria jedno spojenie) a vytvorí sa pole spojení. Jedna položka tohto poľa je pritom ďalšie pole, ktoré má tvar popísaný v tabuľke 5.1.

Index	0	1	2	3	4	5
Popis	Index bodu z ktorého vychádza spojenie	Index bodu do ktorého vchádza spojenie	Index cesty, ktorému spojenie patrí	Vzdialenosť bodov	Časový rozdiel medzi bodmi	Prevýšenie medzi bodmi

Tabuľka 5.1: Formát poľa popisujúceho jedno spojenie

Metóda `Gpx::createConnectionsFromPoints` ešte v závere vytvorí novú inštanciu objektu `Connections`, ktorá podobne ako objekt `Ways` spravuje množinu spojení. Táto inštancia bude návratovou hodnotou spomínanej metódy.

Konštruktor triedy `Gpx` v závere ešte vytvorí inštanciu objektu `NodeFinder`, ktorému parametrami predá množinu bodov a objekty spravujúce množiny spojení a ciest. Tento objekt má za úlohu integrovať novonahrané dáta, čomu je venovaná nasledujúca kapitola.

5.5 Integrácia novej trasy v systéme

Samotná integrácia sa spúšťa metódou `NodeFinder::start`. Jej prvým krokom je spustenie SQL procedúry `SaveNearPoints`, ktorá parametrami dostane interval indexov bodov, pri ktorých sa vyhľadajú ich blízki susedia. Títo susedia budú uložení v databázovej tabuľke `tempNearPoints`. Uložené body spracuje metóda `NodeFinder::getTempNearPoints`, ktorá vytvorí pole dvojíc. Každá takáto dvojica bude mať na prvom indexe index bodu, od ktorého sa blízke body vyhľadávali a na druhom indexe index blízko nájdeného bodu. Týmto spôsobom dostaneme tabuľku blízkych bodov.

Ďalej je potrebné načítať všetky cesty, v ktorých sa blízke body vyskytujú, pretože s najväčšou pravdepodobnosťou budú tieto cesty modifikované. Na to je určená metóda `Ways::loadNearPointsWays`, ktorá získa všetky cesty obsahujúce blízke body. Podobne funguje metóda `Connections::loadNearPointsConnections`, ktorá získa všetky spojenia na cestách, ktoré obsahujú blízke body. Obe tieto metódy pritom vynechávajú cesty a spojenia vytvorené novou trasou, pretože tie už načítané sú.

Z dát potrebných na integráciu novej trasy už chýbajú len body, ktoré neobsahuje

nová trasa, ale nachádzajú sa v tabuľke blízkych bodov. Tieto body získava metóda `NodeFinder::getPointsFromTempTable`.

Teraz sú už všetky potrebné dáta z databázy načítané a môže sa spustiť samotné vyhľadávanie nových uzlov, aktualizácia spojení, ciest a nových bodov. Všetky tieto operácie pokrýva metóda `NodeFinder::processNearPoints`. Veľmi zjednodušene by sa jej obsah dal popísať algoritmom 5.1.

```
1 while tabuľka blízkych bodov nie je prázdna do
2 |   odober prvý záznam z tabuľky blízkych bodov a ulož z neho body do PS_old;
3 |   spriemeruj body v množine PS_old a tento nový bod ulož do newId;
4 |   presuň všetky spojenia a cesty prechádzajúce bodmi v množine PS_old do bodu
  |   newId;
5 |   zmaž body z množiny PS_old;
6 end
```

Algoritmus 5.1: Zjednodušený algoritmus pre spracovanie blízkych bodov

Táto metóda zoberie tabuľku blízkych bodov a pozrie sa, či sa v nej nachádza nejaký záznam. Ak tam taký je, odoberie sa z nej prvý. Do množiny PS_{old} sa uložia indexy bodov nachádzajúce sa v tomto zázname. Pomocou metódy `NodeFinder::averageNearPoints` sa vytvorí nová inštancia bodu, ktorá bude priemerom bodov PS_{old} a uloží sa do zoznamu ostatných bodov pod novým jedinečným indexom $newId$. Riadok 4 algoritmu 5.1 je možné opäť len zjednodušene popísať algoritmom 5.2.

```
1  while množina PS_old nie je prázdna do
2  |   odober index prvého bodu z množiny PS_old a ulož ho do I_old;
3  |   if bod s indexom I_old nie je uzlom then
4  |   |   preruš cestu vedúcu bodom s indexom I_old v tomto bode;
5  |   |   miesto pôvodnej cesty vytvor dve nové s pôvodnými okrajovými bodmi
   |   |   spájajúce sa v bode s indexom newId;
6  |   else
7  |   |   nahraď v cestách použitie indexu bodu I_old za newId;
8  |   end
9  |   nahraď v spojeniach použitie indexu bodu I_old za newId;
10 |   zmaž duplicitné spojenia rovnakých ciest medzi doma bodmi;
11 |   zaktualizuj tabuľku blízkych bodov zmenenými indexami;
12 end
13 zaktualizuj príznak uzla bodu s indexom newId a všetkých bodov, s ktorými tvorí
   spojenie;
14 zmaž všetky duplicitné spojenia medzi bodmi;
15 if bod s indexom newId nemá príznak uzla then
16 |   zaktualizuj príznak uzla bodu s indexom newId;
17 end
```

Algoritmus 5.2: Zjednodušený algoritmus pre aktualizáciu ciest a spojení

Prejde sa teda postupne všetkými bodmi v množine PS_{old} pričom index jedného takéhoto bodu budeme označovať I_{old} a samotný bod P_{old}. Povedzme, že bod P_{old} nemá príznak uzla. Pomocou metódy `Ways::getWaysWithPointIndex` si vyžiadame množinu in-

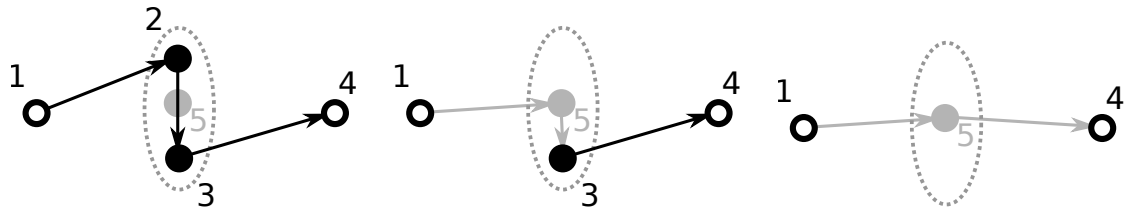
dexov ciest, na ktorých bod P_{old} leží. Ako bolo spomenuté v závere sekcie 4.5, bod, ktorý nemá príznak uzla, môže ležať na jednej alebo dvoch cestách. Každú takúto cestu je potrebné spracovať samostatne. V prvom rade je potrebné získať podrobné informácie o takejto ceste. Na to slúži metóda `Ways::getWayByIndex`, ktorá vráti pole s detailami o zvolenej ceste. Formát takéhoto poľa je v tabuľke 5.2.

Index	0	1	2	3	4
Popis	Index bodu v ktorom cesta začína	Index bodu v ktorom cesta končí	Celková dĺžka cesty	Čas potrebný na prejazd cesty	Celkové prevýšenie cesty

Tabuľka 5.2: Formát poľa popisujúceho jednu cestu

Metódou `Ways::removeWayByIndex` sa teraz táto cesta zmaže, pretože v celku už existovať nebude. Miesto nej sa vytvoria dve nové. Prvá bude vychádzať z pôvodného začiatočného bodu (získaného z detailov cesty) do bodu s indexom $newId$. Ďalej je potrebné zistiť index bodu I_{start}, ktorý predchádza bodu P_{old}. Na to slúži metóda `Connections::getStartPointIndexByEndPointIndexAndWay`. Ak by bol tento index rovný indexu $newId$, bolo by zbytočné vytvárať novú cestu, pretože by vznikla cesta začínajúca a končiaca v bode s indexom $newId$, ktorá by obsahovala iba jedno spojenie.

Príklad 5.5.1 *Na obrázku 5.2 je ukázané, ako môže dôjsť k tomu, že je zbytočné nejakú cestu vytvoriť. Uzly `2` a `3` sa budú zlučovať do nového bodu `5`. Najprv sa zmaže bod `2`, čím dôjde k rozdeleniu cesty `1-4` na cesty `1-5` a `5-4` (obr. 5.2b). Teraz sa má zmazať bod `3` a cesta `5-4` sa má rozdeliť na dve ďalšie. Boli by to cesty `5-5` a `5-4`. Ako je vidieť, prvá z nich nedáva zmysel.*



(a) Počiatočný stav (b) Po odobratí bodu `2` (c) Po odobratí bodu `3`

Obr. 5.2: Postup zlučovania bodov

Obdobne je potrebné získať index bodu I_{end}, ktorý nasleduje za bodom P_{old}. K tomu je určená metóda `Connections::getEndPointIndexByStartPointIndexAndWay`. Opäť je potrebné overiť, či je index $newId$ odlišný od indexu I_{end}.

Po zmene ciest je nutné aktualizovať všetky spojenia nachádzajúce sa na nich. Medzi dvoma bodmi ale môže existovať niekoľko ciest, ktoré prechádzajú rozdielnymi bodmi, takže ako jednoznačne určiť, ktoré spojenia zaktualizovať? Ako riešenie sa ukázalo použiť dve pomocné množiny. Prvou je množina WS_{new} obsahujúca prvky s informáciami o aktualizácii indexov ciest na spojeniach medzi dvoma bodmi s určitým starým indexom cesty. Jedným týmto prvkom je pole, ktorého štruktúra sa nachádza v tabuľke 5.3. Druhou pomocnou množinou je množina CS_{new}, ktorá obsahuje informácie o preznačení indexov ciest v konkrétnych spojeniach. Formát jedného prvku tejto množiny je v tabuľke 5.4.

Index	0	1	2	3
Popis	Nový index cesty	Pôvodný index cesty	Index prvého preznačovaného bodu	Index posledného preznačovaného bodu

Tabuľka 5.3: Formát prvku pomocnej množiny WS_{new}

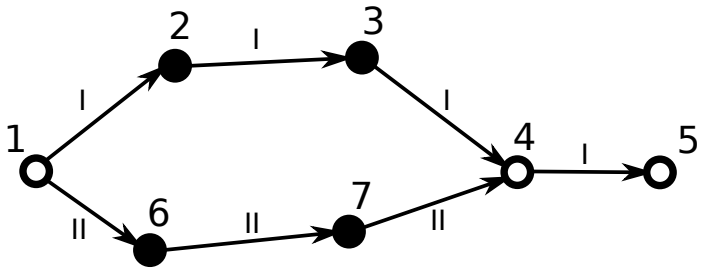
Index	0	1	2
Popis	Nový index cesty	Index bodu, z ktorého vychádza spojenie	Index bodu, do ktorého vchádza spojenie

Tabuľka 5.4: Formát prvku pomocnej množiny CS_{new}

Pri vytváraní novej cesty sa do množiny CS_{new} pridá prvok, ktorý nesie index novej cesty, index bodu *newId* a bod, ktorý je s bodom s indexom *newId* v spojení, čiže buď index I_{start} alebo I_{end}. Do množiny WS_{new} sa potom pridá prvok s indexom novej cesty, indexom cesty, ktorá sa preznačuje, a potom buď index bodu na začiatku pôvodnej cesty s indexom I_{start} alebo index I_{end} s indexom bodu na konci pôvodnej cesty.

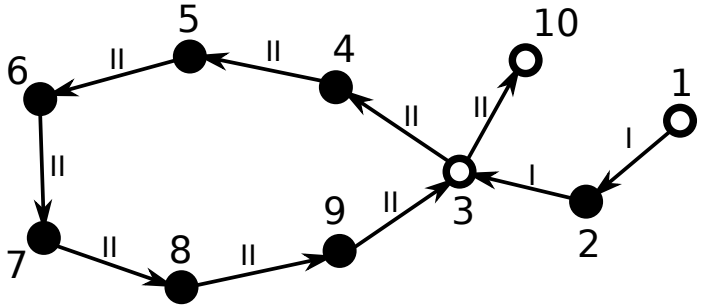
Príklad 5.5.2 *Na obrázku 5.3 je vyobrazený graf s dvoma cestami označenými rímskymi číslicami. Pôvodná cesta* `I` *medzi bodmi* `1` *a* `5` *bola rozdelená v bode* `4`*, čím vznikli cesty* `1-4` *a* `4-5`*. Pôvodná cesta* `I` *týmto zaniká a je potrebné zaktualizovať jej spojenia s novými indexami ciest. Pri vytváraní cesty* `1-4` *sa do množiny CS_{new} pridá prvok* `[III,3,4]`*, čo hovorí, že spojenie medzi bodmi* `3` *a* `4` *bude patriť ceste s indexom* `III`*. Do množiny WS_{new} sa pridá prvok* `[III,I,1,3]`*, čo znamená, že všetky spojenia začínajúce v bode* `1` *s pôvodným indexom cesty* `I` *majú zmeniť index cesty na hodnotu* `III`*, až kým sa nenarazí na bod* `3`*.*

Čo sa týka pridania druhej cesty `4-5`*, tu sa do množiny CS_{new} pridá prvok* `[IV,4,5]`*. Do množiny WS_{new} sa nepridá nič, pretože koncový bod spojenia vychádzajúceho z bodu* `4` *sa rovná koncu aktualizovanej cesty.*



Obr. 5.3: Preznačovanie ciest

Príklad 5.5.3 *Mohlo by sa zdať, že množina CS_{new} je zbytočná a všetko by sa dalo zariadiť pomocou množiny WS_{new}. Na obrázku 5.4 je však znázornená situácia, ktorá sa pri spracovaní reálnych dát vyskytuje pomerne často. Keby sme chceli aktualizovať index cesty* `II` *medzi bodmi* `3` *a* `3`*, nevedeli by sme jednoznačne určiť, ktoré spojenia sa majú preznačiť. Je potrebné začať spojením* `3-4` *alebo* `3-10`*? Keď ale povieme, že spojenie medzi bodmi* `3` *a* `4` *bude patriť ceste* `III`*, a potom všetky spojenia na ceste* `II` *od indexu* `4` *po index* `3` *budú patriť ceste* `III`*, je možné zaktualizovať jednoznačne žiadanú cestu.*



Obr. 5.4: Preznačovanie ciest, ktoré obsahujú sľučku

Pri aplikovaní zmien z množiny CS_{new} sa môže stať, že medzi dvoma bodmi existujú viaceré cesty, čím dôjde k preznačeniu indexov ciest všetkých z nich a vzniknú duplicity. Tento nežiaduci efekt rieši metóda `Connections::removeDuplicitiesAroundPoint`, ktorá tieto duplicity zmaže a ponechá len jedno spojenie.

Následne je potrebné zaktualizovať tabuľku blízkych bodov, na čo bolo poukázané v príkladoch 4.5.4 a 4.5.5. Implementačne je tabuľka blízkych bodov riešená pomocou dvojrozmerného poľa, kde prvý index značí, od ktorého bodu sa vyhľadávalo a druhý index je poradové číslo (počítané od 0) určitého blízkeho bodu. Napríklad v premennej `$nearPoints[25][1]` by sa nachádzal index v poradí druhého blízkeho bodu k bodu s indexom 25. Najprv sa teda skontroluje výskyt indexu I_{old} v prvom indexe tabuľky blízkych bodov. Ak sa tam nájde, naradí sa novým indexom $newId$ a všetky priradené blízke body sa skontrolujú, či sa nachádzajú stále dostatočne blízko, aby boli priemerované. Tie indexy bodov, ktoré sa už dostatočne vzdialili, sa z tohto záznamu vypustia. Po dokončení tejto kontroly sa začne hľadať výskyt indexu I_{old} v hodnotách tabuľky blízkych bodov. Pokiaľ sa taký index nájde, skontroluje sa vzdialenosť medzi bodmi určenými indexami $newId$ a prvým indexom tabuľky blízkych bodov. Ak je vzdialenosť dostatočne blízka, nahradí sa index I_{old} indexom $newId$. Inak sa hodnota z tabuľky zmaže.

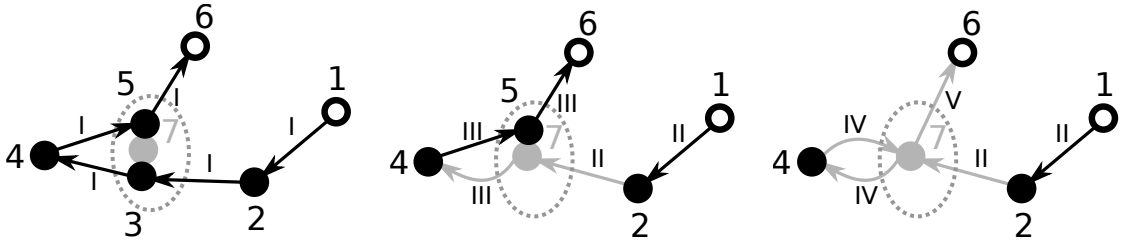
Tento postup sa opakuje, pokiaľ sa neodoberú všetky body z množiny PS_{old}. Teraz prichádza na rad kontrola uzlov bodu s indexom $newId$ a všetkými bodmi, s ktorými tvorí spojenia. Metódou `Connections::createConnectionsMapByStartIndexWithEndIndex` sa získajú spojenia vchádzajúce do zlúčeného bodu s indexom $newId$ a spojenia vychádzajúce z tohto bodu metódou `Connections::createConnectionsMapByEndIndexWithStartIndex`. Obe tieto metódy vracajú pole spojení, ktoré je indexované práve podľa indexu bodu, z ktorého vychádzajú resp. do ktorého vchádzajú. Samotné informácie o spojeniach sa využijú neskôr, teraz však pomocou PHP funkcií `array_keys` a `array_merge` získam množinu indexov bodov, ktoré sa nachádzajú v okolí zlúčeného bodu. Túto množinu ešte nechám prejsť PHP funkciou `array_unique`, ktorá zabezpečí, že všetky indexy sa budú v množine vyskytovať maximálne raz. Do tejto množiny vložím pomocou PHP funkcie `array_unshift` aj index zlúčeného bodu $newId$, aby sa spracoval ako prvý.

Samotné overenie, či má byť bod uzlom vykoná metóda `NodeFinder::isPointNode`. Ak sa zistí, že príznak uzla sa v danom bode zmení, táto zmena sa v bode uloží. Ak bod nebol uzlom a má sa ním stať, skontrolujú sa indexy ciest všetkých spojení v jeho okolí. Spojenia v okolí uzla musia mať vždy jedinečný index cesty, prípadne môže obsahovať cyklickú cestu, čo znamená, že jedno spojenie vychádzajúce z tohto bodu môže mať rovnaký index cesty, ako spojenie vchádzajúce do tohto bodu. Cesta s takýmto indexom však musí v tomto bode začínať a súčasne aj končiť. Ak by sa našli iné duplicity spojení, je treba tieto

cesty rozdeliť a prečíslovať. Túto situáciu demonštruje príklad 5.5.4.

Iný prípad nastáva, ak bod uzlom bol, ale prestal ním byť. Vtedy má vo svojom okolí spojenia s rôznymi indexami ciest (v jednom smere priechodu bodom), čo pri bode, ktorý nie je uzol, nastať nemá. V tejto situácii sa spustí metóda `NodeFinder::checkNotNodePointWays`, ktorá najprv skontroluje bod, z ktorého vychádza spojenie do tohto bodu. Ak ten bod tiež nie je uzlom, zistí sa, aký index cesty má toto spojenie. Je zrejmé, že táto cesta v tomto smere pokračuje ďalej, a preto sa tento index cesty použije aj na spojenia, ktoré nasledujú za bodom, ktorý prestal byť uzlom. Ak sa nepodarí skopírovať index zo spojení pred uzlom, skúsi sa to opačne, teda zistiť, či je nasledujúci bod uzlom a ak nie, uschovať si index cesty spojenia medzi týmito bodmi a tento index použiť pre spojenia pred bodom, ktorý prestal byť uzlom. Prakticky na to poukazuje príklad 5.5.5.

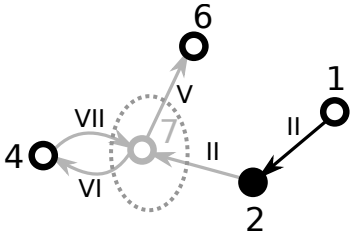
Príklad 5.5.4 *Graf na obrázku 5.5a obsahuje jednu cestu `I` vedúcu z bodu `1` do bodu `6`. V tomto grafe bolo detekované blízke postavenie bodov `3` a `5`. Tieto body je potrebné zlúčiť do nového bodu, v tomto prípade do bodu `7`. Po odobratí bodov `3` a `5` dostaneme graf zobrazený na obrázku 5.5c. Najprv dôjde k testu, či má byť bod `7` vrcholom. Zistí sa že áno (viď sekcia 4.5). Tento bod obsahuje dve spojenia s rovnakým indexom cesty. Cesta s týmto indexom však začína a aj končí v bode `7`, takže cestu nie je potrebné deliť. Následne sa otestuje aj bod `4` a aj tu sa zistí, že tento bod má byť uzlom. Takisto sa tu nájdu dve spojenia s rovnakým indexom cesty, avšak cesta s týmto indexom tu nezačína ani nekončí. Cesta sa preto musí v bode `4` rozdeliť. Výsledné rozdelenie ciest je na obrázku 5.5d.*



(a) Počiatočný stav

(b) Po odobratí bodu `3`

(c) Po odobratí bodu `5`

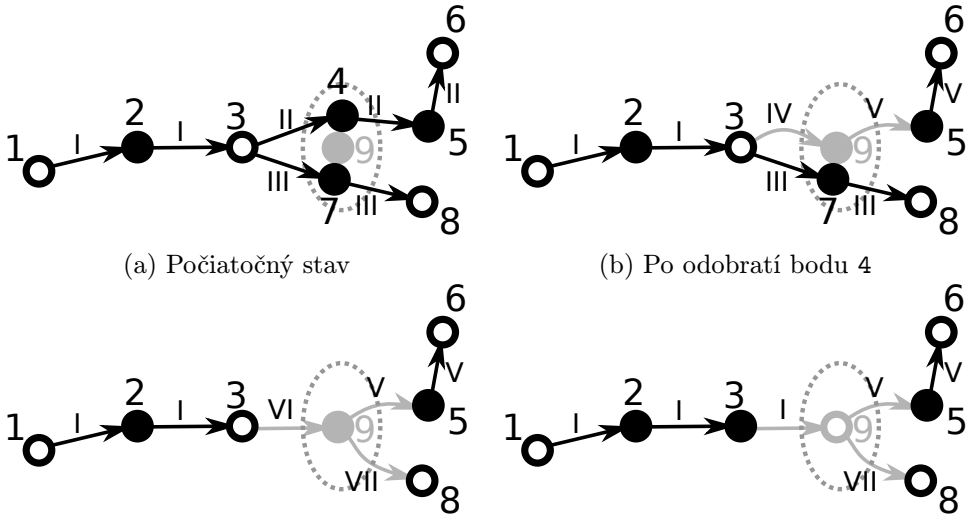


(d) Po kontrole uzlov

Obr. 5.5: Postup zlučovania bodov

Príklad 5.5.5 *Na obrázku 5.6a vidíme graf s troma cestami zbiehajúcimi sa v bode `3`. Body `4` a `7` sa nachádzajú vo svojej blízkosti a budú sa zlučovať. Tieto dva body sa zlúčia a vznikne nový bod `7` (medzi bodmi `3` a `9` by najprv vznikli duplicitné hrany, ktoré sa eliminujú skôr popísaným spôsobom). Stav pred kontrolou príznaku uzlov je na obrázku 5.6c. Príznak uzla sa bude ako prvému kontrolovať zlúčenému bodu. Tu sa pochopiteľne zistí, že tento bod má byť uzlom. Pokračuje sa testovaním bodu `3`, kde sa zistí, že bod má len jedno vchádzajúce spojenie a jedno vychádzajúce, čiže uzlom byť nemá. Pozrie sa teda, či predchádzajúci bod*

(bod `2`) je uzlom. Zistí sa, že nie, a tak sa zoberie index cesty spojenia medzi týmito bodmi (medzi `2` a `3`, čo je `I`), a tento index cesty sa priradí všetkým spojeniam za uzlom `3`, až kým sa nenarazí na bod, ktorý je uzlom. Výsledkom aktualizácie uzlov je obrázok 5.6d.



(a) Počiatočný stav

(b) Po odobratí bodu `4`

(c) Po odobratí bodu `7`

(d) Po kontrole uzlov

Obr. 5.6: Postup vytvárania ciest

Ďalej nasleduje kontrola ciest a spojení medzi bodmi. Nemôže sa totižto stať, že medzi dvoma bodmi budú spojenia patriace viac ako jednej ceste. Možnosť vzniku takýchto spojení je naznačená na obrázku 4.8. S použitím získaných spojení, z ktorých boli skôr získané len indexy okolitých bodov, je teraz možné nájsť viacnásobné spojenia medzi dvoma bodmi. Tieto spojenia sú indexované podľa indexu bodu v okolí a pokiaľ má hodnota (ktorá je tiež pole) viac ako jeden prvok (čo sú konkrétne spojenia), vieme, že sa tu nachádzajú duplicity. Tieto viacnásobné spojenia sa zmažu, spriemerujú sa ich štastistiky (vdzialenosť, čas a prevýšenie) a s týmito hodnotami sa vytvorí nové spojenie.

Na záver, po týchto manipuláciách s hranami sa ešte môže stať, že zlúčenému bodu sa opäť môže zmeniť príznak uzla. Na kontrolu sa opäť použije funkcia `NodeFinder::checkNotNodePointWays`.

Týmto je spracovanie jedného záznamu tabuľky blízkych bodov ukončené. Ak tabuľka neobsahuje už žiadne ďalšie záznamy, práca s pridávaním novej trasy do systému je hotová a vykonané zmeny sa môžu uložiť do databázy.

5.6 Urýchlenie vykonávania zmien v databáze

V predchádzajúcich sekciách neboli zmeny v databáze zámerne vykonávané priebežne, ale až po dokončení všetkých operácií. Ide o to, že jednotlivé požiadavky na databázu sú veľmi „drahé“ a nakoľko jedno pridanie trasy pracuje s tisíckami záznamov v tabuľke, nie je mysliteľné vykonávať tieto zmeny priebežne.

Pri vytváraní nového spojenia pomocou metódy `Connections::createNewConnection` nedôjde k okamžitému uloženiu záznamu do databázy. Vytvorí sa len nový záznam v objekte, ktorému sa priradí zatiaľ nepoužitý identifikátor (používajú sa záporné čísla, ktoré iné spojenia s istotou nemajú). Po zavolaní metódy `Connections::commit` dôjde k vytvoreniu

jednej SQL požiadavky obsahujúcej všetky nové vytvorené spojenia. Rovnako aj pri mazaní záznamov, tie sa tiež nemažú okamžite. Len sa poznačia indexy, ktoré sa na záver zmažú (ak je index väčší ako 0) alebo sa len odoberú zo zoznamu na pridanie do databázy (ak je index menší ako 0). Ani modifikácia záznamov sa nevykonáva okamžite, taktiež sa ukladá zoznam indexov záznamov, ktoré sa majú na záver zaktualizovať. Po zavolaní metódy `Connections::commit` sa ešte vytvorí rozdiel množín záznamov na modifikáciu a na zmazanie, aby sa zbytočne nemodifikovalo niečo, čo sa aj tak zmaže. Týmto spôsobom sa miesto tisícok požiadaviek vykonajú len tri, čo má vysoký dopad na zrýchlenie práce s databázou.

Tri množiny pre správu záznamov (vytvorenie nových, modifikáciu a zmazanie) podobne využívajú aj objekty `Ways` (správa ciest) a `NodeFinder` (správa bodov). Líšia sa ale pri vytváraní nových záznamov. Pri spojeniach bol index záznamu nepodstatný, pretože sa tento identifikátor v iných záznamov nepoužíva. Označenia bodov a ciest sa naopak využívajú veľmi často. Pri bodoch sa to dá vyriešiť tak, že poznáme približný počet bodov, ktoré sa vytvoria vďaka počtu záznamov v tabuľke blízkych bodov. Zlučovaním istej množiny bodov vždy vznikne nový. Vytvorí sa preto v tabuľke daný počet záznamov, zistíme si ich indexy a tie potom používame. Ak sa stane, že nejaký záznam z tabuľky bodov vypadne (niektoré body sa postupným zlučovaním od seba vzdialili), pridá sa index nového bodu do množiny indexov na zmazanie. Zvyšné budú v zozname indexov na modifikáciu (pretože na začiatku mal záznam len nulové hodnoty). Síce sa takto vytvoria nejaké záznamy zbytočne (v tomto prípade je ich minimum), má to však dopad na ďalšie zrýchlenie práce s databázou.

Pri správe ciest je to s vytváraním nových záznamov o niečo komplikovanejšie, pretože vopred nepoznáme počet ciest, ktoré bude potrebné vytvoriť. Vytvorí sa preto zvolený počet nových záznamov určený konštantou `Ways::MYSQL_INSERT_BUFFER` rovnou 50. Keď sa vyčerpajú všetky záznamy a bude potrebné vytvoriť ďalšiu cestu, vytvorí sa ďalších 50 záznamov. Na záver sa nepoužité cesty zmažú a použité zaktualizujú. Tu sa síce riadkami plytvá o niečo viac, než pri bodoch, ale má to cenu zrýchlenia behu skriptu a databáza disponuje dostatkom voľných indexov.

Databáza má pri každom bode uloženú pozíciu pomocou zemepisnej šírky a zemepisnej dĺžky. Ako bolo spomenuté v sekcii 2.5, body je potrebné zlúčiť, ak sa nachádzajú vo vzdialenosti menšej než 15 metrov. Databáza samotná však nemá k dispozícii funkciu, ktorá by dokázala nájsť body v určtej vzdialenosti určenej metrickými jednotkami. Funkcie sa samozrejme dajú vytvárať aj dodatočne, avšak vyhľadávať v tisícoch záznamov body vzdialené od seba menej ako 15 metrov pomocou funkcie počítajúcej vzdialenosť bodov na povrchu gule je veľmi zdĺhavé. Funkcia počítajúca vzdialenosť bola preto optimalizovaná na počítanie vzdialenosti bodov v rovine. Tento výpočet síce nie je úplne presný, ale na požiadavky tejto aplikácie plne postačujúci. Nepredpokladá sa, že by sa podobná sieť ciest vytvárala v okolí zemských pólov. Zemepisná dĺžka môže byť teda uvedená v intervale $< -180; 180 >$ a zemepisná šírka v intervale $< -90; 90 >$. Keď sa pozícia šírky vynásobí dvoma, dostávame štvorec, v ktorom sa už vzdialenosť bodov vyhľadáva pomerne jednoducho. Akurát je potrebné prepočítať a aproximovať 15 metrov na rozdiel v stupňoch, ale to už nie je problém.

5.7 Vyhľadávanie cesty

Samotné vyhľadávanie ciest je oddelené od užívateľského rozhrania umožňujúce výber bodov a zobrazovania výslednej trasy. Vyhľadávanie je spracované v súbore `/search.php`

a očakáva tri parametre prijaté metódou `POST`:

- `start` - čiarkou oddelená zemepisná dĺžka a zemepisná šírka štartovného bodu v stupňoch
- `end` - čiarkou oddelená zemepisná dĺžka a zemepisná šírka koncového bodu v stupňoch
- `type` - typ vyhľadávania, môže byť (reťazec):
  - `distance` - vyhľadávanie najkratšej cesty
  - `time` - vyhľadávanie najrýchlejšej cesty

Vyhľadávanie na základe informácií z databázy vykonáva objekt `WayFinder`. Konštruktor tohto objektu vyžaduje súradnice bodov, medzi ktorými sa bude vyhľadávať. Metódou `WayFinder::setSearchingType` sa nastaví, podľa ktorého ohodnotenia hrán sa bude vyhľadávať. Môže to byť jedna z konštant `WayFinder::SHORTEST_DISTANCE` (vyhľadávanie najkratšej cesty) alebo `WayFinder::SHORTEST_TIME` (vyhľadávanie najrýchlejšej cesty). Pred spustením vyhľadávania je ešte možné metódou `WayFinder::setTwoWaySearching` nastaviť, či sa bude pri vyhľadávaní cesty zanedbávať orientácia hrán (nastavením hodnoty `true`). Vyhľadávanie cesty sa potom spustí metódou `WayFinder::start`.

Tu sa v prvom rade zistí ku indexom ktorých bodov sú zvolené body najbližšie. Aj pre štartovný aj pre cieľový bod sa zavolá metóda `WayFinder::getWaysOfPoint`, ktorá zistí index najbližšieho bodu a množinu ciest, na ktorých sa daný bod nachádza. Ďalej nasleduje vyhľadanie spojení, ktoré sa nachádzajú na nájdených cestách metódou `WayFinder::getConnectionsOnWays`. Pomocou týchto dát je možné vytvoriť pomocné cesty medzi zvolenými bodmi a najbližšími bodmi podobne ako to bolo naznačené na obrázku 4.11b. Štartovací bod obdrží dočasný index uzlu s hodnotou -1 a koncový bod index -2. Ďalej je potrebné získať údaje o všetkých bodoch, ktoré sú uzlami a o vzťahoch medzi nimi (o cestách). Metóda `WayFinder::initialiseNodes` načíta všetky body, ktoré majú príznak uzlov (potrebné sú predovšetkým ich indexy a pozície) a zoznam ciest z databázy načíta metóda `WayFinder::initialiseEdges`. Vchádzajúce resp. vychádzajúce cesty sú indexované podľa indexu bodu, do ktorého vchádzajú resp. z neho vychádzajú cesty. Formát jedného takéhoto záznamu sa nachádza v tabuľke 5.5.

Index	**Popis**
0	Index bodu na druhom konci cesty
1	Vzdialenosť medzi bodmi na ceste
2	Časová vzdialenosť bodov na ceste
3	Prevýšenie cesty
4	Index tejto cesty

Tabuľka 5.5: Formát uchovávania informácií o ceste

Ďalej nasleduje kontrola, či vyhľadávané body neležia na jednej ceste, pretože v takom prípade by sa cesta nemusela vyhľadávať cez ostatné uzly. Ak sa zistí, že je umožnené zanedbať orientáciu hrán alebo sa body nechádzajú v smere jazdy vytvorí sa ešte jedna hrana medzi štartovným a koncovým bodom.

Teraz prichádza na rad metóda `WayFinder::searchWay`, ktorá sa pokúsi nájsť cestu medzi zvolenými bodmi. Informácia o každom bode je uchovaná v štruktúre popísanej

Index	Popis
0	Označenie uzlu – môže byť `UNMARKED` (neoznačený), `OPENED` (otvorený) alebo `CLOSED` (uzvaretý)
1	Vzdialenosť (čas), ktorú(-ý) je nutné pri zatiaľ vyhľadanej ceste nutné prekonať
2	Odhadovaná vzdialenosť (čas) do cieľa
3	Pole súradnic tohto bodu (`[0]` – zem. dĺžka, `[1]` – zem. šírka)
4	Index predchádzajúceho bodu na zatiaľ nájdenej ceste
5	Index cesty, ktorou sa podarilo nájsť zatiaľ najlepšiu cestu do tohto bodu
6	Ak sa sem podarilo nájsť cestu v správnom smere `true`, inak `false`

Tabuľka 5.6: Formát uchovávania informácií o uzle

v tabuľke 5.6. Informácie sú zvolené z dôvodu umožnenia vyhľadávania cesty pomocou A* algoritmu popísanom v sekcii 3.5.

Vyhľadávanie začína tým, že sa nastaví uzol so štartovným indexom cesty ako otvorený, prejdená vzdialenosť (čas) sa vynuluje a vypočíta sa odhadovaná vzdialenosť (čas) do cieľa. Odhadovaná vzdialenosť sa počíta pomocou vzdialenosti vzdušnou čiarou, čas taktiež pomocou vzdialenosti vzdušnou čiarou podelenú zvolenou priemernou rýchlosťou nastavenou v konštante `WayFinder::AVERAGE_SPEED`. Ide len o heuristickú pomocnú funkciu, takže nemusí byť úplne presná. Táto konštanta je nastavená na hodnotu 5 m/s (18 km/h). Nakoniec sa do množiny otvorených uzlov pridá index štartovného bodu. Týmto sa spustí vyhľadávanie najkratšej cesty pomocou algoritmu popísanom v sekcii 3.5. Ak sa náhodou stane, že v množine otvorených bodov nebude žiadny index a cieľový bod stále nebol dosiahnutý, cesta medzi bodmi neexistuje a nastaví sa chybová hodnota na konštantu `WayFinder::NO_WAY_FOUND`.

Pri nájdení cesty sa na záver metódou `WayFinder::createFinalPoints` získa množina všetkých bodov na ceste v poradí od štartovného bodu po koncový. Pri získavaní súradníc bodov sa ide od cieľového bodu. V tom je možné zistiť, pomocou akej cesty sme sa do tohto bodu dostali. Tým pádom je možné zistiť aj všetky spojenia na tejto ceste a ich príslušné body. Každou cestou sa pri tom môže ísť oboma smermi, takže uložený smer je potrebné zohľadniť a získavať spojenia cesty zo správneho konca. Až sa postupne dopracujeme do štartovného bodu, body celej cesty sú uložené a hľadanie cesty je dokončené.

Na záver je už len postačujúce spustiť metódu `WayFinder::getStats`, ktorá vráti štatistiky o ceste, vrátane bodov, vo formáte JSON. Príkladom výstupu môže byť napríklad:

```
{
        error: 0,
        points: [
                {latitude: 49.0752, longitude: 16.5789},
                {latitude: ... },
                ...
        ],
        distance: 1.235,
        time: 321.43,
        elevation: 17.32
}
```

Parameter `time` sa vygeneruje iba v prípade hľadania najrýchlejšej cesty. Pri hľadaní najkratšej cesty sa zanedbáva orientácia hrán a v prípade nájdenia cesty s hranou s opačnou

orientáciou nie je možné odhadnúť dobu jazdy. Parameter `error` obsahuje nulovú hodnotu v prípade, že sa cesta nájde, inak obsahuje hodnotu 1. Vzdialenosť určená parametrom `distance` je udávaná v kilometroch a parameter `time` udáva potrebnú dobu jazdy v sekundách. Parameter `points` obsahuje kolekciu bodov zoradenú od štartovného bodu po cieľový. Prvý a posledný bod pritom nepatrí priamo vyhľadanej ceste, ale bodom, ktoré vyžiadal užívateľ.

5.8 Užívateľské rozhranie pre vyhľadávanie trás

Interaktívne užívateľské rozhranie umožňujúce zadávanie štartovacieho a cieľového bodu a samotné vyhľadávanie sa nachádza v súbore `/index.php`. Pre zobrazenie mapy, značiek vybraných bodov a výslednej cesty sa používa Mapy.cz API, popísané v sekcii 5.2.

Užívateľ si najskôr zvolí ľubovoľné dva body, medzi ktorými chce cestu vyhľadávať. Tieto body potom môže presúvať a súčasné pozície sa zakaždým ukladajú do premenných `startLocation` a `endLocation` v skriptovacom jazyku JavaScript. Po kliknutí na jedno z vyhľadávacích tlačidiel sa spustí JavaScript-ová funkcia `searchWay`, ktorá použije uložené súradnice bodov a spustí požiadavku na súbor `search.php` s využitím technológie AJAX. Odpoveďou tejto požiadavky je JSON reťazec, ktorý obsahuje všetky potrebné dáta pre zobrazenie cesty. Štatistiky o nájdenej trase sa skopíruju na pre užívateľa viditeľné miesta a o prekreslenie cesty sa postará funkcia `redrawPath`.

Kapitola 6

Testovanie

Po vyhľadaní cesty pomocou navrhnutej a implementovanej aplikácie dostaneme vzdialenosť udávajúcu určitú hodnotu. Je ale potrebné overiť, či sú tieto čísla reálne a či odrážajú skutočnú vzdialenosť medzi dvoma bodmi.

Na overovanie získaných výsledkov som sa rozhodol použiť webovú aplikáciu Google Maps. Pri testovaní ale bolo potrebné dávať pozor na výber trasy, pretože cyklisti často používajú trasy mimo hlavných ciest a v cestnej sieti zas poznal Google Maps kratšie trasy, než boli cyklistami namerané.

Najprv som do systému nahral iba jednu trasu, pretože na začiatok je potrebné overiť, či priemerovanie bodov, tvorba spojení a ciest a hlavne výpočet vzdialeností pomocou GPS súradníc pracuje správne. Tabuľka 6.1 zobrazuje začiatočný a koncový bod a vzdialenosti namerané vypracovanou aplikáciou a aplikáciou Google Maps.

Štartovný bod	**Cieľový bod**	**Aplikácia**	**Google Maps**
$16,5652800°E$ $49,2639539°N$	$16,5751933°E$ $49,2659983°N$	$0,914\,km$	$0,9\,km$
$16,5824569°E$ $49,2280728°N$	$16,5703011°E$ $49,2362767°N$	$1,331\,km$	$1,3\,km$
$16,5190386°E$ $49,2323606°N$	$16,5351803°E$ $49,2210939°N$	$2,059\,km$	$2,1\,km$

Tabuľka 6.1: Porovnanie vzdialeností pri použití jednej trasy

Štartovný bod	**Cieľový bod**	**Aplikácia**	**Google Maps**
$16,5231264°E$ $49,2284583°N$	$16,5214847°E$ $49,2254314°N$	$0,802\,km$	$0,8\,km$
$16,5763628°E$ $49,2195839°N$	$16,5765131°E$ $49,2257922°N$	$0,977\,km$	$1,0\,km$
$16,5368486°E$ $49.2207647°N$	$16,5531672°E$ $49.2180175°N$	$1,303\,km$	$1,3\,km$

Tabuľka 6.2: Porovnanie vzdialeností pri použití dvoch trás

Druhým krokom pri testovaní bolo pridanie ďalšej trasy do systému. Bolo potrebné overiť či zlučovanie bodov pri vytváraní uzlov nedeformuje vypočítané vzdialenosti medzi

spojeniami. Tabuľka 6.2 obsahuje namerané hodnoty pri hľadaní najkratších ciest cez aspoň jeden uzol.

Na záver boli do systému pridané štyri trasy a cesty boli vyhľadávané cez čo najväčší počet uzlov. Tabuľka 6.3 zobrazuje namerané výsledky.

Štartovný bod	**Cieľový bod**	**Aplikácia**	**Google Maps**
$16,5346814°E$ $49,2211781°N$	$16,5940761°E$ $49,2355061°N$	$7,552\,km$	$7,2\,km$
$16.5652156°E$ $49.2640519°N$	$16.6059636°E$ $49.3109389°N$	$8,913\,km$	$9,3\,km$
$16.6630411°E$ $49.2958278°N$	$16.5926600°E$ $49.2352819°N$	$13,548\,km$	$13,7\,km$

Tabuľka 6.3: Porovnanie vzdialeností pri použití štyroch trás

Výsledky vyhľadávaných ciest sú dostačujúco presvedčivé, že výpočet vzdialeností funguje správne. Občasné odchýlky vo vzdialenostiach sú spôsobené predovšetkým tým, že porovnávať trasy medzi zvolenými aplikáciami nie je úplne možné. Vždy sa nájde nejaký úsek cesty, ktorý jedna z aplikácii nepozná.

Vyhľadaný čas jazdy medzi dvoma bodmi nie je možné medzi týmito dvoma aplikáciami porovnať, keďže Google Maps nepozná v okolí Brna cyklistické trasy. Avšak nakoľko je možné tvrdiť, že vzdialenosti sú pomerne presné a vypočítaná priemerná rýchlosť odpovedá možnej rýchlosti jazdy na bicykli, je pravdepodobné, že zobrazená doba jazdy je správna. Príklady nameraných výsledkov je možné vidieť v tabuľke 6.4.

Štartovný bod	**Cieľový bod**	**Vzdialenosť**	**Čas**	**Rýchlosť**	**Prevýšenie**
$16.5817917°E$ $49.2288997°N$	$16.5782833°E$ $49.2302097°N$	$0,277\,km$	$37\,s.$	$27,15$ km/h	$0,321\,m$
$16.5905572°E$ $49.2552575°N$	$16.6270353°E$ $49.2643881°N$	$3,806\,km$	$20\,min.$ $51\,s.$	$10,95$ km/h	$202,723\,m$
$16.4394950°E$ $49.2429314°N$	$16.5690136°E$ $49.2119906°N$	$10,838\,km$	$38\,min.$ $8\,s.$	$17,05$ km/h	$199,129\,m$
$16.6418839°E$ $49.2450047°N$	$16.5659236°E$ $49.2619794°N$	$11,134\,km$	$43\,min.$ $8\,s.$	$15,49$ km/h	$262,996\,m$

Tabuľka 6.4: Porovnanie získaných časov na základe vzdialenosti a prevýšenia

Podobne aj overenie vyhľadávania najkratšej cesty vychádza z toho, že vzdialenosti sú počítané správne. Testovaním aplikácie bolo overené, že postupným vzďaľovaním cieľového bodu od štartovného došlo pri zobrazovaní nájdenej cesty pri určitej hranici k „prepnutiu“ na inú vetvu cesty, pričom vzdialenosť v tomto momente zostala takmer nezmenená. Posúvaním bodu potom po tejto novej vetve sa zas vzdialenosť zmenšovala, čo značilo, že hľadanie najkratšej cesty je správne.

Kapitola 7

Záver

Cieľom tejto práce bolo vytvoriť aplikáciu umožňujúcu vyhľadávať najvhodnejšiu trasu nad údajmi zozbieraných z reálneho prostredia. Ďalším cieľom bolo navrhnúť aplikáciu s čo najjednoduchším užívateľským rozhraním, pričom by si zachovala plnú funkčnosť.

Výsledkom tejto práce je funkčná aplikácia, ktorej implementácia bola popísaná. Nie je možné však povedať, že je dokonalá. Naráža predovšetkým na potrebnú dobu behu pri pridávaní novej trasy, ktorá sa síce podarila niektorými optimalizáciami skrátiť, stále sa však počíta v jednotkách minút a pribúdajúcim počtom ciest (najmä pri väčšom prekrytí existujúcich a nových ciest) stále stúpa. Väčšina konfigurácií PHP interpretov má pritom nastavenú maximálnu dobu behu na 30 sekúnd, takže toto nastavenie je potrebné pred používaním aplikácie zmeniť. Problémom je taktiež množstvo potrebnej pamäte predovšetkým pri pridávaní novej trasy, ale pri vyššom počte uložených trás aj pri vyhľadávaní cesty. Pri pridávaní ani nie desiatky trás nemá aplikácia problém „zhltnúť“ necelý GB pamäte. Do budúcna je tu preto istá cesta pre optimalizácie.

Pri nie vysokom počte trás aplikácia pracuje bezchybne. Blízke body sa správne zlučujú, cesty získavajú správne indexy, čo potom umožňuje rýchle vyhľadávanie ciest. Vzdialenosť nájdených trás medzi dvoma bodmi je na základe testovania taktiež bezchybná a v ostatných údajoch sa očividne taktiež nevyskytujú chyby.

Pri pridávaní väčšieho počtu trás sa však objavili problémy pravdepodobne s indexovaniami ciest. Chyba zrejme nebude veľká avšak jej zdroj sa len pri veľkom počte dát pomerne ťažko vyhľadáva. Je však prioritou číslo jedna túto chybu odhaliť a opraviť.

Konečné vyhľadávanie ciest sa podarilo dotiahnuť takmer podľa predstáv. Väčšina vyhľadávaných ciest sa nájde do jednej sekundy, občas sa však objaví oneskorenie do 2-3 sekúnd. V budúcnosti by zrejme bolo výhodnejšie použiť aspoň vylepšenú variantu algoritmu A*, ktorá by prehľadávala graf z oboch koncov, alebo použiť iný výkonnejší algoritmus.

V ďalšom vývoji tejto aplikácie by taktiež bolo vhodné zoptimalizovať vytváranie nových ciest. Nemuselo by sa to riešiť najjednoduchším spôsobom zrušenia jednej cesty a vytvorením dvoch nových, ale rušením len jednej a modifikáciou druhej. Prípadne by bolo možné miesto vytvárania nových záznamov použiť záznamy s indexami určenými na zrušenie. Implementácia tried s týmto návrhom počítala, avšak implementovať ho by bolo vhodné až po úplnom odladení aplikácie, nakoľko by sa sťazilo ladenie skriptu.

Aj vyhľadávanie ciest by sa dalo zoptimalizovať. Pri veľkom počte uložených trás by to bola dokonca nutnosť, nakoľko v tejto verzii aplikácie sa pred vyhľadávaním načítajú všetky uzly a všetky spojenia. Tých môžu byť neskôr tisíce, čo by výrazne spomalilo vyhľadávanie. Bude preto potrebné nájsť spôsob, ktorý nenačíta všetky záznamy naraz, ale postupne, pričom zas bude potrebné klásť dôraz na minimálny počet požiadaviek na data-

bázu.

Nemenej dôležitou súčasťou budúcej aplikácie by mala byť aj správa existujúcich trás. Teraz je možné pridať do systému dve rovnaké trasy, čo síce nijak výrazne aplikácii neuškodí, avšak všetky údaje by tým získali nesprávne vyššie váhy a neskoršie zlučovanie informácií by tak bolo ovplyvnené. Detekcia rovnakých trás by pritom mohla využiť informácie uložené v hlavičke GPX súboru, alebo skontrolovať pozície prvých niekoľkých bodov.

Posledným navrhovaným vylepšením do budúcnosti by mohlo byť upravenie zobrazovanej nájdenej trasy viacerými farbami. Momentálne sa používa iba červená (modrá je len pomocná). Farba by v budúcnosti mohla rozlišovať náročnosť jednotlivých úsekov a miesta, kde vzniká najväčšie prevýšenie.

Literatúra

[1] Foster, D.: GPX 1.1 Schema Documentation. http://www.topografix.com/GPX/1/1/, [cit. 2015-01-09].

[2] Foster, D.: GPX: the GPS Exchange format. http://www.topografix.com/gpx.asp, [cit. 2015-05-24].

[3] Google: KML Reference - Keyhole Markup Language – Google Developers. https://developers.google.com/kml/documentation/kmlreference, [cit. 2015-05-24].

[4] Google: KML Tutorial - Keyhole Markup Language – Google Developers. https://developers.google.com/kml/documentation/kml_tut, [cit. 2015-05-24].

[5] Kika, O.: BIKELAND. http://www.horskakola.wz.cz/, [cit. 2015-01-09].

[6] Majer, T.: *Problémy trasovania v rozsiahlych dopravných sieťach*. Dizertační práce, Žilinská Univerzita v Žiline, 2013.

[7] Mio™: O technologii GPS. http://eu.mio.com/cs_cz/global-positioning-system_presnost-gps-a-zdroje-chyb.htm, [cit. 2015-01-09].

[8] Palúch, S.: *Algoritmická teória grafov*. Žilinská Univerzita v Žiline, 2008.

[9] Pašák, M.: Presnosť GPS. http://www.maclab.sk/clanky/presnost-gps.php, [cit. 2015-01-13].

[10] Seznam.cz: API Mapy.cz. http://api4.mapy.cz/, [cit. 2015-01-13].

[11] The PHP Group: PHP: What is PHP? http://php.net/manual/en/intro-whatis.php, [cit. 2015-01-13].

[12] Česáková, J.; Křížová, M.: Metodická příručka pro práci s přístrojem GPS. http://hsh.uhk.cz/files/vystupy/ka5/Metodick%C3%A9%20pokyny%20pro%20tvorbu%20tras/Metodick%C3%A9_pokyny_pro_tvorbu_tras_s_pomoc%C3%AD_GPS.pdf, [cit. 2015-01-13].

Dodatok A

Obsah CD

Priložené CD obsahuje nasledujúcu štruktúru priečinkov:

Cesta	**Popis**
aplikácia/gpx súbory/	Pár GPX súborov pre možné nahranie do aplikácie.
aplikácia/zdrojové súbory/	Zdrojové súbory potrebné na beh aplikácie.
technická správa/latex/	Zdrojové texty tohto dokumentu napísané v LaTeXe.
technická správa/pdf/	Tento dokument v elektronickej podobe.

Tabuľka A.1: Štruktúra priečinkov